НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Аналітично-інформаційна система створення критичних медичних заявок»

Виконав:
студент IV курсу, групи КП-62
Самойленко Назарій Юрійович _____

Керівник:
Старший викладач кафедри ПЗКС, к.т.н.,
Хіцко Яна Володимирівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри СПіСКС, к.т.н.,
Боярінова Юлія Євгенівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Самойленку Назарію Юрійовичу

1. Тема проєкту «Аналітично-інформаційна система створення критичних медичних заявок», керівник проекту старший викладач кафедри ПЗКС, к.т.н. Хіцко Я.В, затверджені наказом по університету від «25» травня 2020 р. № 1181-с

2. Термін подання студентом проєкту «16» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз мов програмування та технологій розроблення веб-додатків;
   - розроблення архітектури застосунку;
   - опис розроблених алгоритмів та модулів системи;
   - аналіз розробленого веб-додатку.

5. Перелік обов'язкового графічного матеріалу:
   - життєвий цикл заявки з критично необхідного інвентаря. Схема алгоритму (креслення);
   - діаграма розподілу прав доступу користувача (креслення);

- архітектура веб-додатку (плакат);
- структура веб-сторінок (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	14.11.2019	
2.	Розроблення та узгодження технічного завдання	28.11.2019	
3.	Розроблення структури веб-додатку	15.12.2019	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2019	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2020	
6.	Підготовка матеріалів другого розділу дипломного проєкту	20.02.2020	
7.	Програмна реалізація web-ресурсу	10.03.2020	
8.	Тестування веб-додатку	17.03.2020	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2020	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	11.04.2020	
11.	Підготовка графічної частини дипломного проєкту	21.04.2020	
12.	Оформлення документації дипломного проєкту	26.05.2020	

Студент Назарій САМОЙЛЕНКО

Керівник проєкту Яна ХІЦКО

АНОТАЦІЯ

Дипломний проєкт присвячено розробці веб-додатку для створення заявок з критично необхідного інвентаря. В проєкті спроєктовано та реалізовано повний життєвий цикл заявки від створення до завершення виконання.

Веб-додаток являє собою клієнт-серверний застосунок, розроблений з використанням сучасних засобів для розробки веб-застосунків. Інформаційна безпека сервісу реалізована за рахунок розподілу прав доступу користувача. Користувачі мають доступ лише до функціоналу, що передбачений їх роллю в системі. Система має відкритий API, що може працювати з зовнішніми системами. За допомогою графічного інтерфейсу будь-який користувач може створити заявку з критично необхідного інвентаря чи заявку на додавання нового інвентаря до існуючої бази даних. Користувач з правами доступу адміністратора також має можливість переглядати статистичні дані, що накопилися в ході роботи системи. Архітектура системи дозволяє додатку легко розширюватися додаючи нові функціональні можливості.

У даному дипломному проєкті розроблено: архітектуру веб-сервісу, клієнтську та серверну частину, базу даних, алгоритм створення та опрацювання заявок з критично необхідного інвентаря та заявок на додавання нового інвентаря в базу даних.

ABSTRACT

Diploma project dedicated to developing web application for creating requests for the critically necessary equipment. The project was designed and implemented the full life cycle of the application from creation to completion.

Web application is a client-server application, developed with modern technologies and best practices. Information security of the service is implemented through the distribution of user access rights. Users have access only to the functionality provided by their role in the system. The system has an open API that can work with external systems. Using the graphical interface, any user can create a request from critical inventory or a request to add new inventory to an existing database. The user with administrator rights also can view statistics accumulated during the operation of the system. The system architecture allows the application to be easily expanded by adding new functionality.

In this diploma project, the following is developed: web service architecture, client and server part, database, the algorithm of creation and processing of requests from critical inventory, and requests for adding new inventory to the database.

АННОТАЦИЯ

Дипломный проект посвящен разработке веб приложения для создания заявок с критически необходимого инвентаря. В проекте спроектировано и реализовано полный жизненный цикл заявки от создания до завершения исполнения.

Веб приложение представляет собой клиент-серверное приложение, разработанное с использованием современных средств для разработки веб-приложений. Информационная безопасность сервиса реализована за счет распределения прав доступа пользователя. Пользователи имеют доступ только к функционалу, предусмотренным их ролью в системе. Система имеет открытый API, который может работать с внешними системами. С помощью графического интерфейса любой пользователь может создать заявку с критически необходимого инвентаря или заявку на добавление нового инвентаря к существующей базы данных. Пользователь с правами администратора также имеет возможность просматривать статистические данные, накопившиеся в ходе работы системы. Архитектура системы позволяет приложению легко расширяться, добавляя новый функционал.

В данном дипломном проекте разработано: архитектуру веб-сервиса, клиентскую и серверную часть, базу данных, алгоритм создания и обработки заявок с критически необходимого инвентаря и заявок на добавление нового инвентаря в базу данных.

ДП.045440-01-90 Аналітично-інформаційна система створення критичних медичних заявок. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Аналітично-інформаційна	4	
	система створення критичних		
	медичних заявок. Технічне		
	завдання		
ДП.045440-03-81	Аналітично-інформаційна	54	
	система створення критичних		
	медичних заявок. Пояснювальна		
	записка		
ДП.045440-04-51	Аналітично-інформаційна	4	
	система створення критичних		
	медичних заявок. Програма та		
	методика тестування		
ДП.045440-05-34	Аналітично-інформаційна	10	
	система створення критичних		
	медичних заявок. Керівництво		
	користувача		
ДП.045440-06-99	Аналітично-інформаційна	1	
	система створення критичних		
	медичних заявок. Життєвий		
	цикл заявки з критично		
	необхідного інвентаря. Схема		
	алгоритму		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Аналітично-інформаційна	1	
	система створення критичних		
	медичних заявок. Діаграма		
	розподілу прав доступу		
	користувачів		
ДП.045440-08-98	Аналітично-інформаційна	1	
	система створення критичних		
	медичних заявок. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

АНАЛІТИЧНО-ІНФОРМАЦІЙНА СИСТЕМА СТВОРЕННЯ КРИТИЧНИХ МЕДИЧНИХ ЗАЯВОК

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:
_____ Яна ХІЦКО

Нормоконтроль:
_____ Микола ОНАЙ

Виконавець:
_____ Назарій САМОЙЛЕНКО

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Аналітично-інформаційна система створення критичних медичних заявок.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка присвячена створенню аналітично-інформаційної системи для створення заявок з критично необхідного інвентаря. Система дозволяє створити заявку та пройти весь життєвий цикл до завершення.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Веб-сервіс повинен забезпечувати такі основні функції:

- авторизація користувачів у системі;
- розподіл прав доступу користувачів у системі;
- створення заявки з критично необхідного інвентарю та заявок на додавання нового інвентарю;

- опрацювання заявки з критично необхідного інвентаря та заявок на додавання нового інвентарю;
- пошук інвентаря за багатьма характеристиками.

Розробку виконати на мові TypeScript з використанням технологій Angular, Nest.js та СКБД PostgreSQL.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
    - «Життєвий цикл заявки з критично необхідного інвентаря. Схема алгоритму»;
    - «Діаграма розподілу прав доступу користувачів».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту 12.11.2019
Розроблення та узгодження технічного завдання 25.11.2019
Розроблення структури веб-додатку 12.12.2019
Підготовка матеріалів першого розділу проєкту 23.12.2019
Розроблення дизайну сторінок та графічних елементів 08.02.2020
Підготовка матеріалів другого розділу проєкту 17.02.2020
Програмна реалізація веб-додатку 13.03.2020
Тестування веб-додатку 22.03.2020
Підготовка матеріалів третього розділу проєкту 08.04.2020
Підготовка матеріалів четвертого розділу проєкту 26.04.2020
Підготовка матеріалів графічної частини проєкту 13.05.2020
Оформлення технічної документації проєкту 26.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

АНАЛІТИЧНО-ІНФОРМАЦІЙНА СИСТЕМА СТВОРЕННЯ КРИТИЧНИХ МЕДИЧНИХ ЗАЯВОК

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:
_____ Яна ХІЦКО

Нормоконтроль:
_____ Микола ОНАЙ

Виконавець:
_____ Назарій САМОЙЛЕНКО

2020

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ERP – Enterprise Resource Planning, організаційна стратегія інтеграції виробництва і операцій, управління трудовими ресурсами, орієнтована на безперервне балансування і оптимізацію ресурсів підприємства за допомогою спеціалізованого інтегрованого пакета прикладного програмного забезпечення.

НСЗУ – Національна Служба Здоров'я України.

HTTP – HyperText Transfer Protocol, протокол прикладного рівня передачі даних. Основою є технологія «клієнт-сервер».

TCP – Transmission Control Protocol, один з основних протоколів передачі даних в інтернеті, призначений для управління передачею даних.

UDP – User Datagram Protocol, протокол передачі даних в мережі інтернет.

TLS – Transport Layer Security, протокол захисту транспортного рівня, криптографічний протокол, що забезпечує захищену передачу даних між вузлами в мережі інтернет.

DNS – Domain Name System, комп'ютерна розподілена система для отримання інформації про домени. Використовується для отримання IP-адреси по імені хоста.

XML – eXtensible Markup Language, розширювана мова розмітки.

JSON – текстовий формат обміну даних, заснований на мові програмування JavaScript. JSON легко читається людьми.

AJAX – Asynchronous JavaScript and XML, технологія звернення до сервера без перезавантаження сторінки.

SQL – Structured Query Language, декларативна мова програмування, що застосовується для створення, модифікації та управління даними в реляційній базі даних, керованої відповідною системою управління базами даних.

API – Application Programming Interface, опис з способів (набір класів, процедур, функцій, структур або констант), якими одна комп'ютерна програма може взаємодіяти з іншою програмою.

SOLID – принципи об’єктно – орієнтованого програмування та орієнтування, описані Робертом Мартіном на початку 2000-х років. Включають в себе 5 правил: single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion.

MVC – Model-View-Controller, архітектури побудови програмного забезпечення, що полягає в розділені даних застосунку, користувацького інтерфейсу та бізнес – логіки.

ORM – Object Relational Mapping, технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування.

IDE – Integrated Development Environment, система програмних засобів, яка використовується програмістами для розробки програмного забезпечення;

NPM – менеджер пакетів, що входить до складу Node.js.

CLI – Command Line Interface, інтерфейс командного рядка.

DOM – Document Object Model, це незалежний від платформи і мови програмний інтерфейс, що дозволяє програмам і скриптам отримати доступ до вмісту HTML-, XHTML- і XML-документів, а також змінювати, структуру і оформлення таких документів.

HTML – HyperText Markup Language, мова розмітки гіперктекстових документів.

ACID – Atomicity, Consistency, Isolation, Durability (набір властивостей, що гарантують надійну роботу транзакцій бази даних: атомарність, узгодженість, ізольованість, довговічність).

Фреймворк – інфраструктура програмних рішень, що полегшує розробку складних систем.

Вендор – це компанія, яка виробляє і / або поставляє товари і послуги під власним брендом.

ВСТУП

Сучасний світ поступово трансформується з аналогового в цифровий. Трансформуються майже всі сфери нашого життя, бізнес, медіа та навіть державний бюрократичний апарат. Зараз для того щоб виконати певні операції, які раніше потребували годин стояння в чергах, треба витратити лише пару хвилин скориставшись онлайн сервісом. Країни з високим економічним розвитком «діджиталізували» майже всі сфери свого державного апарату, Україна також намагається слідувати тенденціям, та вже має приклади успішних проєктів. Сьогоденний розвиток технологій дозволяє працювати з великими об’ємами даних, в кожної людини є принаймні смартфон з доступом в інтернет, цифровий світ став реальністю. Разом з тим сучасний світ кидає нові виклики для вирішення яких можливо ефективно використовувати інформаційні системи.

Прикладом такої сфери є медицина. Інформаційні технології могли б значно покращити рівень послуг та покращити роботу сфери загалом. Даний проєкт є концептом, який зможе спростити деякі формальні процедури, створити певну інформаційну систему яку потім можна буде розвивати та додати нові можливості

Ситуація з короновірусною пандемією показує наскільки важливим є швидкість прийняття рішень, інформаційна система допоможе консолідувати інформацію та пришвидшити бюрократичні процеси.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз існуючих програмних рішень

1.1.1. eZdorovya

eZdorovya – це розробник технічного ядра eHealth в Україні. Система намагається вирішити широкий спектр проблем сфери охорони здоров'я. Основними завданнями є:

- забезпечення прозорості фінансування системи охорони здоров'я;
- поступовий перехід на електронний обіг та можливість працювати без паперу (е-картка, е-рецепт, е-направлення);
- формування бізнес середовища для створення нових електроних сервісів;
- створення простору для інновацій в медицині (машине навчання, великі дані, блокчейн тощо).

Система складається з центральної бази даних та електронних медичних інформаційних систем які підключаються до неї. Міністерство охорони здоров'я регулює впровадження eHealth на нормативно-правовому рівні. Зараз до ядра підключена низка інформаційних систем, а саме:

- програмний комлекс Аптека – програма для автоматизації товарообігу в аптеках;
- Helsi – інформаційна система для взаємодії закладів вторинної та третинної ланки з електроною системою охорони здоров'я та роботи з електронними направленнями від сімейних лікарів;
- Medics – інформаційна система для знаходження та запису на прийом до лікаря. Система передбачає персональну сторінку кожного лікаря з відгуками та рейтингом.

Це лише декілька прикладів, з більше ніж тридцяти підключених наразі. Як бачимо всі вони виконують певні аналітично – інформаційні функції. В наступному розділі більш детально розглянемо одну з таких.

1.1.2. SimplexMix

Комплексна інформаційно-медична система автоматизації закладів охорони здоров'я. Є ERP-системою та охоплює всі ключові процеси діяльності лікарні. Складається з таких модулів як:

- поліклініка;
- стаціонар;
- лабораторія.

Також виділенні окремі модулі адміністрування такі як: Склад, Персонал, Аналітика, Фінанси. Система має різні тарифні плани та пропонує різні пакети: *стартовий* – містить весь необхідний функціонал для роботи з eHealth та НСЗУ та є безкоштовною, *базовий* – забезпечує повну інформатизацію клінічної частини діяльності закладу та коштує від 400 грн/роб. Місце за місяць, *повний* – ключова складова системи планування та управління ресурсами закладу ERP.

1.1.3. Висновки

У даному підрозділі були розглянуті існуючі інформаційні системи охорони здоров'я України. Як бачимо в ході реформи, цифрова трансформація відіграє провідну роль, більшість інформаційних систем створені для автоматизації роботи лікарні з пацієнтами, що не конкурує з нашою системою.

1.2. Аналіз існуючих технологій

1.2.1. Клієнт-серверна архітектура

Клієнт-серверна архітектура є основою розробки веб-додатків різних масштабів. Кінцевий користувач за допомогою клієнтського застосунку може взаємодіяти з серверним, який в свою чергу представляє свої ресурси у виді даних. Взаємодія клієнта і сервера відбувається за рахунок мережевих протоколів, таких як: HTTP, TCP, UDP, TLS за допомогою останнього можливо налаштувати безпечну передачу даних. Сервера можуть

розміщуватися на різних машинах утворюючи кластери. Це дає змогу балансувати навантаження між серверами за допомогою проксі - серверів, прикладом таких є nginx та Apache HTTP Server [1]. Для збільшення продуктивності можливе збільшення кількості екземплярів сервера при збільшенні навантаження від клієнтів. За правильність з'єднання клієнтського та серверного застосунку відповідаю DNS сервери. Також зараз багато хмарних технологій, які дозволяють придбати в них всю необхідну інфраструктуру та запускати серверний код, також вони пропонують свої рішення в балансуванні навантажень, захисту та зберіганню даних, тощо, найбільш популярними серед таких є Amazon Web Services, Microsoft Azure, Google Cloud Computing [2]. Сервер взаємодіє з базами даних, які також можуть розміщуватись на різних машинах, зараз популярним рішенням є контейнеризація застосунків за допомогою такого програмного забезпечення як Docker в незалежні ізольовані контейнери на рівні операційної системи та подальша їх оркестрація. Серверні застосунки мають бути відмово стійкими та в разі збільшення навантаження масштабуватись.

В застосунку клієнтський застосунок слугує графічним інтерфейсом користувача за допомогою якого він взаємодіє з інформацією що міститься в системі. За допомогою протоколу HTTP клієнт надсилає запити до сервера та отримує у відповідь необхідні дані. Для передачі даних використовуються певні текстові формати такі як XML чи JSON. З розвитком веб-технологій технологія взаємодії клієнту і серверу постійно покращувались, якщо на початку розвитку для кожного такого запиту потрібно було перезавантажувати сторінку, то з появою технології AJAX це стало відбуватись асинхронно та не помітно для користувача. Також до клієнтських застосунків є певні вимоги щодо зручності роботи кінцевого користувача, при роботі в веб-браузері сторінки мають завантажуватися швидко, інтерфейс має бути інтуїтивно зрозумілим та доступним для людей з різними фізичними обмеженнями.

1.2.2. Бази даних для зберігання інформації

Для збереження даних існують два типи баз даних – це реляційні бази даних та NoSQL рішення.

Реляційні бази даних в основі використовують таблицю як основну структуру даних, тоді як NoSQL рішення використовують колекції документів в яких дані можуть мати різнорідну форму. Кожен з підходів має свої недоліки та переваги, які слід використовувати для зберігання тих чи інших даних [3]. Головною перевагою реляційних баз є гарантія несуперечливості даних при дотримані форм нормалізації. Для швидкого пошуку використовуються спеціальні структури – індекси, що являють собою бінарне дерево пошуку, індекси створюються полів, пошук по яким відбувається найчастіше. Коли застосунок росте та додаються нові властивості, малоймовірно, що структура бази даних (схеми його таблиць) залишаться не змінними. В такому випадку приходиться змінювати структуру даних в таблиці, для цього використовують сценарії міграції, що автоматично оновляють схему і перетворюють існуючі дані. Для роботи з реляційними базами даних використовується мова запитів SQL, різні реалізації можуть мати свій певний діалект. Зазвичай такі бази даних використовуються для зберігання чітко структурованих даних, які не часто змінюються. Можна виділити декілька головних недоліків реляційних баз даних та випадків коли їх застосування не є доречним. Оскільки дотримання несуперечності даних вимагає нормалізації бази даних за допомогою внутрішніх та зовнішніх ключів, то для отримання певної сутності потрібно використовувати операції JOIN використання яких є дорогим за часом та ресурсами, отже коли потрібно працювати з великими об’ємами даних, використання реляційних баз даних може призвести до повільної роботи системи. З цього можна зробити висновок, що реляційні бази даних не дуже вдалий вибір для роботи з великими об’ємами даних, та з не чітко структурованими даними [3].

Натомість нереляційна модель не використовує табличних зав’язків. Вона майже ніколи не потребує з’єднувати інформації з декількох записів. Найбільш відомим типом NoSQL баз даних є документі сховища, в них записи зберігаються в такому виді в якому вони потрібні застосунку. Нереляційних модель передбачає можливість дублювання інформації при необхідності. Однак дубльовані дані складно своєчасно оновлювати і підтримувати їх несуперечливість. З іншого боку, групуючи відповідні дані, документе сховище може запропонувати більшу гнучкість:

- не потрібно об’єднувати таблиці;
- можливо не використовувати фіксовані схеми;
- кожен запис може мати власні поля.

Серед NoSQL також баз даних виділяють [4]:

- сховища «ключ-значення», які здебільшого використовують для кешування даних;
- графові бази даних, записи в яких зберігаються у виді вершин, а зв’язки у виді ребер.

Нереляційні бази даних є чудовим вибором в роботі з великими об’ємами даних чи з роботою з не чіткими даними.

Отже, можна підсумувати, що реляційні БД орієнтовані на дані: вони максимізують структурування даних і усувають їх дублювання незалежно від того, в якому вигляді ті потрібні. Нереляційні БД, навпаки, орієнтовані на застосування: вони полегшують доступ до даних і їх використання відповідно до ваших потреб. NoSQL дозволяють швидко і ефективно зберігати великі, мінливі і неструктуровані дані. Не турбуючись про фіксовані схеми і міграції, ви можете розробляти свої рішення набагато швидше. Проте потрібно пам’ятати, що відповідальність за оновлення дубльованої інформації по всіх документах та колекціях лежить лише на користувачу таких БД. Було прийнято рішення використовувати реляційні бази даних для розробки даного проєкту, а саме PostgreSql, так як структура даних буде чіткої та такою, що легко приводиться до однієї з нормальних форм.

1.3. Висновки

У даному розділі було розглянуто базові технології, на основі яких буде будуватися архітектура системи. Технологія «Клієнт-сервер» є галузевим стандартом та безальтернативним вибором при побудові веб-застосунків на сьогодні. Серверна частина буде складатися з монолітного серверу, що використовує реляційну базу даних PostgreSql для зберігання даних.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Обґрунтування вибору мови програмування для серверної частини

2.1.1. Огляд мови програмування Java

Java – строго типізована об’єктно-орієнтована мова загального призначення розроблена компанією Sun Microsystems. Основною ідеєю застосування мови є принцип «написано один раз, працює в будь-якому місці» (WORA), що означає, що скомпільований код Java може працювати на всіх платформах, які підтримують Java, без необхідності перекомпіляції. Програми Java компілюються у байт-код, який потім може виконуватися на будь-якому комп’ютері на якому встановлено віртуальну машину JVM [5]. Java є однією з найпопулярніших мов для розробки програмного забезпечення корпораційного рівня, її використовують для створення веб-серверів, мобільних та десктопних додатків. Всередині Java існує декілька основних сімейств технологій:

- Java SE – Java Standard Edition, основне видання Java, включає компілятори, API, Java Runtime Environment, підходить для створення користувацьких додатків;
- Java EE – Java Enterprise Edition, являє собою набор специфікацій для створення програмного забезпечення корпоративного рівня. В 2017 році проєкт був переданий Eclipse Foundation, після чого був перейменований на Jakarta EE. Модулі Java EE видалені з Java SE починаючи з 11 версії;
- Java ME – Java Micro Edition, створена для використання на пристроях обмежених по обчислюваній потужності.

Мова Java використовується для створення мобільних застосунків під популярну операційну платформу Android. Основною перевагою використання Java є об’єктно-орієнтований підхід закладений в Java за дизайном. Під час використання мови легко дотримуватися

принципів SOLID та об’єктно-орієнтованого дизайну [6]. Java має велику спільноту користувачів та багато перевірених часом рішень. Так як дипломний проєкт є саме веб-додатком, цікаво буде оцінити можливості Java саме для створення веб-серверів. Найбільш популярним є рішенням Spring – універсальний фреймворк з відкритим кодом для Java платформи. Він надає велику свободу Java – розробникам в проєктуванні систем, крім того він надає добре задокументовані та легкі у використані засоби вирішення проблем корпоративного рівня [7]. Spring може розглядатися як колекція менших фреймворків, а саме:

- Inversion of Control (IOC) контейнер конфігурування компонентів застосунку та управління життєвим циклом Java-об’єктів;
- фреймворк доступу до даних працює з системами управління реляційними базами даних на Java-платформі, використовує JDBC та ORM засоби для вирішення поставлених задач;
- фреймворк MVC – каркас оснований на HTTP та сервлетах, надає багато можливостей для розширення та налаштування;
- фреймворк аутентифікації та авторизації – інструментарій процесів аутентифікації та авторизації, що підтримує багато популярних стандартів та підходів, протоколів та практик через проєкт Spring Security.

Отже можна зробити висновок, Java є чудовим вибором та індустріальним стандартом для створення програмного забезпечення корпоративного рівня, вона має багато підходів та хороших практик, які розробники програмного забезпечення можуть використовувати у своїх програмах. Мова має свого вендора та спільноту користувачів, що покращує її розвиток та підтримку. Нові релізи почали відбуватися регулярно, спільнота знає коли відбудеться наступний реліз. До мінусів можна віднести, дещо повільнішу швидкість роботи, так як Java, використовує віртуальну машину та байт-код, а не компілюється у бінарний код, який потім запускається, як це роблять C та C++.

2.1.2. Огляд мови програмування Python

Python – скриптова інтерпретована мова загального використання з сильною динамічною типізацією. Мова була вперше випущена в 1991 році та відрізняється від інших популярних мов своїм синтаксисом. В Python не використовується С-подібний синтаксис, натомість використовує значні пробіли для кожної строки коду. Це, за задумкою автора мови Гвідо ван Россума, має підвищувати читабельність коду. Мова активно використовується для створення веб-серверів (популярні фреймворки: Flask та Django), в машинному навчанні та науці про даних [8]. Python має автоматичне керування пам'яттю та підтримує декілька парадигм програмування, включаючи об'єктно-орієнтоване, функціональне та процедурне програмування. Серед головних переваг Python можна виділити:

- низький поріг входження для початку написання програм на мові Python;
- кросплатформеність – програмний код може виконуватись на будь-якій машині;
- велика стандартна бібліотека;
- велика спільнота користувачів;
- простий та виразний синтаксис;
- велика кількість бібліотек для вирішення проблем в різних галузях, це як і веб-програмування так і машинне навчання та наука про даних, особливо зручним Python є для вирішення математичних проблем [8].

Також слід визначити ряд певних недоліків мови програмування Python:

- використання динамічної типізації – відсутня перевірка типів на етапі компіляції, внаслідок цього можливе виникнення помилок під час роботи програми [9];
- низька швидкодія програмного коду, так як, мова є інтерпретованою та виконується на віртуальній машині [10];

- специфічна реалізація об’єктно-орієнтованого програмування, а саме відсутність інтерфейсів у дизайні мови;
- неможливість перезавантаження методів на етапі компіляції.

Для розробки веб -серверів існує два популярних фреймворки: Django та Flask. Django – веб-фреймворк з відкритим кодом, що використовує в основі шаблон MVC, має багато вбудованих можливостей, таких як: робота з базами даних, використовує власну ORM в якій на основі моделей описаних за допомогою Python генеруються відповідні схеми бази даних.

Flask – мінімалістичний фреймворк, що містить мінімальний каркас та надає лише базові функції для створення веб-застосунків. Використовує шаблонізатор Jinja2 для створення користувацьких шаблонів. Отже можна зробити висновок, що Python популярне рішення для створення веб-додатків, яке має багато користувачів, проте мова має декілька особливостей реалізації, а саме відсутність статичної типізації, що не є зручним для деяких розробників програмного забезпечення.

2.1.3. Огляд мови програмування TypeScript та платформи Node.js

TypeScript – мова програмування представлена компанією Microsoft в 2012 році і є засобом програмування веб-застосунків, що значно розширює можливості мови JavaScript, а саме додає статичну типізацію, при цьому використовуючи двигун JavaScript для виконання коду [11]. Головним архітектором мови є Андерс Хейлсберг, що також приймав участь у створенні мови C#. TypeScript є обернено сумісним з JavaScript, що означає що будь-який дійсний код JavaScript, виконається в середовищі TypeScript. Наявність статичної типізації призване підвищити надійність розробки, та полегшити проведення рефакторингу та перевикористання коду, у TypeScript, деякі помилки можна відслідкувати на етапі компіляції. Мова має широку підтримку в різних редакторах для написання коду та IDE, за рахунок власного language server [12]. Мова підтримує використання об’єктно-орієнтованого програмування, а саме має можливість використання

класів, інтерфейсів та абстрактних класів. Головною перевагою мови є можливість написання як клієнтського так і серверного застосунку. Популярною платформою для створення серверних застосунків на мові TypeScript є платформа Node.js – програмна платформа основана на двигуні V8, що транслює JavaScript у машинний код. Платформа розроблена за допомогою мови програмування C++, та у своїй основі має асинхронне програмування та введення і виведення. Node.js є молодою технологією, вона вийшла як експериментальна технологія лише в 2009 році. Для роботи з бібліотеками інших користувачів використовуються пакетні менеджери: npm або yarn. Серед головних переваг Node.js та Typescript можна виділити:

- використання статичної типізації – можливість перевірки типів на етапі компіляції;
- велика спільнота користувачів, яка включає в себе як і розробників серверної частини (Backend інженерів), так і розробників клієнтської частини (Frontend інженерів);
- можливість написання як і клієнтської так і серверної частини;
- Node.js є асинхронним по замовчуванню, що означає, що при написані застосунків використовуються не блокуючі операції, що дозволяє обробляти декілька операцій у фоновому режимі;
- наявність великого вендора – Microsoft. Компанія Microsoft є справжнім технологічним гігантом, її інженери є основними контриб'юторами проєкту. Проте, так як TypeScript, як і Node.js є технологіями з відкритим програмним код, будь хто може внести свою зміну в проєкти.

Серед недоліків варто зазначити наступне:

- Програмний код виконується двигуном JavaScript, який має слабку типізацію, отже результат виконання може бути дещо несподіваним в деяких випадках.
- Так як технологія відносно молода, на сьогодні існує не дуже багато рішень для проєктів корпоративного рівня.

- Сховище для зберігання пакетів npm, є слабо контролюємо, що дозволяє завантажувати туди будь який програмний код, без відповідної на те рецензії. Даний код може містити критичні вразливості з інформаційної безпеки.

Слід окремо розглянути веб-фреймворк для створення веб-серверів Nestjs. Це прогресивне рішення з відкритим програмним кодом для створення ефективних, надійних та масштабованих веб-серверів. У своїй основі він намагається використовувати найкращі практики об’єктно-орієнтованого дизайну та принципів SOLID [6], що є індустріальним стандартом створення програмного забезпечення. Він має свій CLI (інтерфейс командної строки), що пришвидшує написання програмного коду. Крім цього Nestjs має вбудовані можливості для роботи з наступними технологіями:

- модуль для роботи з популярною мовою запитів GraphQL;
- модуль для роботи з протоколом миттєвого обміну повідомленнями – web sockets;
- модулі для роботи з реляційними та NoSQL базами даних;
- модуль для роботи з мікросервісами;
- модуль для роботи з Swagger, що дає можливість використовувати можливості OpenApi для генерації коду на клієнтській частині.

Отже, можна зробити висновок, що TypeScript та Node.js молоді та стрімко набираючі популярність інструменти для створення веб-серверів, які мають широку підтримку серед спільноти інженерів та великих компаній.

2.1.4. Висновки

Проаналізувавши різні мови та підходи до створення веб-серверів, було вибрано мову TypeScript та платформу Node.js для створення серверної частини застосунку. Головним критерієм вибору став факт, що клієнтська частина буде також писатися на мові TypeScript, та наявність у фреймворку Nestjs засобів генерації коду для клієнтської частини.

2.2. Обґрунтування вибору фреймворку для клієнтської частини

2.2.1. Angular

Angular – веб-фреймворк з відкритим кодом для стоворення клієнтської частини веб-додатку. Написаний на мові програмування TypeScript компанією Google, а також іншими сторонніми розробниками. Представляє собою універсальний інструмент для всіх платформ, а саме: веб-браузера, мобільного чи десктопного застосунку. Angular також підтримує серверний рендеринг, що дозволяє створювати швидкі та продуктивні застосунки. В дизайн фреймворку закладено використання принципів об'єктно-орієнтованого дизайну, а саме ін'єкції залежностей (dependency injection, DI) та реактивного програмування [13], бібліотека Rx.js. Angular має гарну документацію, та підтримку майже всіма редакторами коду та IDE. Фреймворк вміщує в себе все необхідне для побудови клієнтського застосунку корпоративного рівня:

- модульну архітектуру, яка за використання ін'єкції залежностей дозволяє створювати архітектуру з слабкою зв'язністю компонентів;
- широкий спектр можливостей для створення та управління життєвим циклом компонентів;
- вбудований роутер для навігації між сторінками додатку з підтримкою лінивого завантаження окремих модулів;
- набір кращих практик, для написання та розвертання додатку для кінцевого користувача;
- використовує за замовчуванням TypeScript для розробки програмного забезпечення.

Серед недоліків можна виділити великий розмір кінцевого застосунку, хоча в останній версії фреймворку, було представлено новий двигун для рендерингу Ivy, що значно покращив ці показники. Для менеджменту стану застосунку популярним рішенням є використання бібліотеки NgRx, що реалізує шаблон flux у комбінації з реактивним програмуванням.

Angular є вдалим вибором для великих корпоративних проєктів, які розраховані на тривалий термін, він є чітко структурованим та дозволяє легше інтегрувати в роботу нових членів команди. Компанія Google використовує Angular у своїх популярних продуктах, таких як: Gmail, YouTube, Google photos тощо.

2.2.2. React

React – JavaScript бібліотека для створення користувацьких інтерфейсів розроблена компанією Facebook. React використовує декларативний підхід [14] до опису користувацьких інтерфейсів, та, на відміну від Angular є бібліотекою, а не фреймворком. Це означає, що React надає лише базовий функціонал для створення клієнтських застосунків. Його можна використовувати для створення як і одно сторінкових (Single page application) так і мобільних застосунків, за використання бібліотеки React Native.

В своїй роботі React використовує віртуальний DOM [15], та створює кеш – структуру, що дозволяє обчислювати різницю між минулим та майбутнім станом інтерфейсу, для оптимального оновлення браузера. Таким чином бібліотека сама вирішує які частини сторінки оновлювати, тим самим підвищуючи продуктивність. React наслідує функціональній парадигмі програмування [14], та використовує композицію функцій на перевагу наслідування класів. Ця філософія істотно відрізняє його від того ж Angular, що базується на принципах об’єктно-орієнтованого програмування.

Також істотною різницею є використання шаблонного синтаксису JSX (або ж TSX для TypeScript) – це розширений синтаксис JavaScript, який дозволяє використовувати схожий на HTML синтаксис для опису структури інтерфейсу. React використовує одно направлений потік даних, що дозволяє уникати багатьох помилок при оновлені стану компонентів. За допомогою бібліотеки були розроблені такі популярні продукти компанії Facebook, як веб-сайт Facebook та Instagram.

React містить широкий спектр різноманітних бібліотек, в своїй екосистемі, що в деяких випадках є як і перевагою так і недоліком.

2.2.3. Vue

Vue – фрейворк з відкритим кодом, для створення користувацьких інтерфейсів, написаний співробітником Google Еваном Ю у 2014 році. Головною метою було створення фреймворку для швидкого прототипування складних користувацьких інтерфейсів. На той час React ще був лише на початкових стадіях розробки, у популярними були такі складні фреймворки як Angular.js та Backbone.js орієнтовані на MVC архітектуру.

Для заповнення цього пробілу і був створений Vue.js, який потім з інструмента швидкого прототипування переріс в інструмент повноцінної розробки. Vue не є монолітним фреймворком, та може легко підключатися до існуючого проєкту. Він, як і Angular, має інтерфейс командного рядка (CLI) для швидкого створення шаблонних компонентів, бібліотеку для управління станом застосунку – Vuex, та гарну підтримку серед майже всіх редакторів коду та IDE. Головним користувачем фрейворку є компанія Alibaba, також у своїй розробці його використовує компанія GitLab. Також Vue підтримує використання TypeScript, що покращує його використання у великих проєктах.

2.2.4. Висновки

Проаналізувавши три найпопулярніших фреймворки для створення клієнтської частини, було вирішено використовувати Angular під час розробки дипломного проєкту. Він є спорідненим з Nestjs який буде використовуватись для розробки серверної частини (розробники Nestjs надихалися Angular під час створення фреймворку) та використовує схожі принципи об’єктно-орієнтованого підходу.

2.3. Обґрунтування вибору системи керування базами даних

2.3.1. Вибір між реляційною та NoSQL базою даних

Головним у виборі бази даних для виконання дипломної роботи було визначити який з двох підходів до зберігання даних буде більш вдалим для проєкту. В розділі 1.2.2 частково було проаналізовано бази даних та виділено переваги та недоліки кожного з підходів. Проаналізуємо дані які будуть зберігатися в базі даних. Основними схемами у базі даних будуть наступні:

- інвентар – більшість полів будуть стандартизовані та визначені, проте можуть існувати деякі властивості характерні для специфічного інвентаря;
- лікарня – всі поля стандартизовані та попередньо визначені;
- лікар;
- обробник заявок.

Як бачимо структури мають стандартизовані та попередньо визначені поля, для зберігання специфічних даних для інвентаря можливо використовувати json, що підтримується у багатьох реляційних СКБД (зокрема в PostgreSQL). Іншим фактором є об’єм даних що має зберігатись в базі даних, заявок планується не велика кількість, інвентарю може бути достатньо багато записів, проте не наскільки, щоб можна було говорити про поняття Big Data [16]. Отже, можна зробити висновок, що використання реляційних СКБД є більш доцільним для дипломної роботи. В наступних розділах розглянемо найбільш популярної реляційні бази даних такі як PostgreSQL та MySQL.

2.3.2. PostgreSQL

PostgreSQL – об’єктно-орієнтована реляційна система управління базами даних. Базується на мові запитів SQL. Не має обмежень на розмір бази даних, та має дуже великий максимальний розмір таблиці – 32 терабайти та поля – 1 гігабайт. Має надійні та високопродуктивні механізми транзакцій та реплікацій. Підтримує слабко структуровані дані у форматі JSON з

можливістю їх індексації [17]. Має широку підтримку індексів, а саме підтримуються наступні типи індексів: B-tree, хеш, GiST, GIN, BRIN, Bloom. При необхідності можливо створювати свої типи індексів.

PostgreSQL підтримує велику кількість вбудованих типів даних: числові типи, символьні типи довільної довжини, двійкові типи (включаючи BLOB), типи для роботи з датами та часом, бульові (логічні) типи, перерахування, геометричні примітиви, мережеві типи, UUID-ідентифікатори, XML дані, масиви, JSON та ідентифікатори об’єктів в базі даних. PostgreSQL дотримується принципів ACID (атомарність, послідовність, ізоляцію, довговічність). Слід зазначити, що PostgreSQL пропонує три рівні ізоляції транзакцій: Read Commited, Repeatable Read та Serializable [18]. Оскільки PostgreSQL не захищена від брудних зчитувань, запит на читання нездійсненної транзакції забезпечує виконання цієї транзакції.

PostgreSQL підтримує повну серійну здатність за допомогою технології ізоляції з серійними знімками (SSI). PostgreSQL дітримує всі функції роботи з реляційними базами даних, та пропонує частину своїх, наприклад, широкий функціонал для роботи з географічними даними. Слід зазначити, що PostgreSQL має вбудовані засоби для повнотекстового пошуку, тому при використані даної СКБД не потрібно використовувати допоміжні інструменти для пошуку, наприклад ElasticSearch. Для цього в PostgreSQL існує спеціальний тип даних tsvector, що являє собою нормалізовану рядок, за яким буде проходити пошук. Під нормалізацією розуміється відкидання стоп-слів, таких, як прийменники, обрізання закінчень слів, тощо.

2.3.3. MySQL

MySQL – реляційна система керування базами даних, розробку та підтримку здійснює корпорація Oracle. Продукт розповсюджується як і під ліцензією GNU General Public License, так і під власною комерційною ліцензією, крім того розробники створюють функціональність по

замовленню ліцензованих користувачів. Спільнотою розробників MySQL було створено декілька відгалуджень, найбільш популярним з яких є MariaDB. MySQL має власний API, та багато комплектів для розробки програмного забезпечення [19] для різних мов програмування, таких як: C++, Java, C#, Python. Цікаво відмітити наявність таблиць типу Maria – розширених версій сховища MyISAM, з додаванням засобів збереження цілісності даних після краху. У разі краху проводиться відміна результатів виконання поточної операції або повернення до певного стану [20]. Якщо порівнювати MySQL з Oracle DB, остання здебільшого використовується для великих корпоративних проєктів, тоді як MySQL, більш гнучка та використовується меншими компаніями. MySQL підтримує інтерфейс ODBC (Open Data Bases Connectivity, відкрита взаємодія з базами даних) може використовуватись для одночасного з'єднання з різними СКБД. Основним з недоліків MySQL є низька швидкість роботи сховища даних, та використання власних правил роботи з NULL значеннями та зі значеннями по замовчуванню. Проте, не зважаючи на ці недоліки, MySQL, має велику популярність та багато користувачів.

2.4. Висновки

Отже проаналізувавши дані, що будуть зберігатися в системі та рішення, які пропонують реляційні та NoSQL бази даних було вирішено використовувати реляційні. Далі для вибору конкретної СКБД було проаналізовано дві найпопулярніші бази даних: PostgreSQL та MySQL. В результаті аналізу було прийняте рішення використовувати PostgreSQL для використання в дипломній роботі, головною перевагою було наявність вбудованого повнотекстового пошуку, що буде використовуватись для пошуку інвентаря.

3. ХАРАКТЕРИСТИКИ РОЗРОБЛЮВАНОЇ СИСТЕМИ

3.1. Моделювання системи та алгоритми

Дипломна робозробка буде складатися з трьох частин: клієнтський застосунок, серверний застосунок, сховища даних для медіа та основної бази даних. В системі передбачено наявність трьох ролей користувачів: обробники заявок, адміністратори та медичні працівники. Відповідно до цього, маємо дерево функціональності, що наведено на рис. 3.1, що може робити кожна з ролей. Ролі є динамічними, що означає, що певний користувач може мати одночасно декілька ролей, це означає, що в такому випадку функціональність користувача додається.

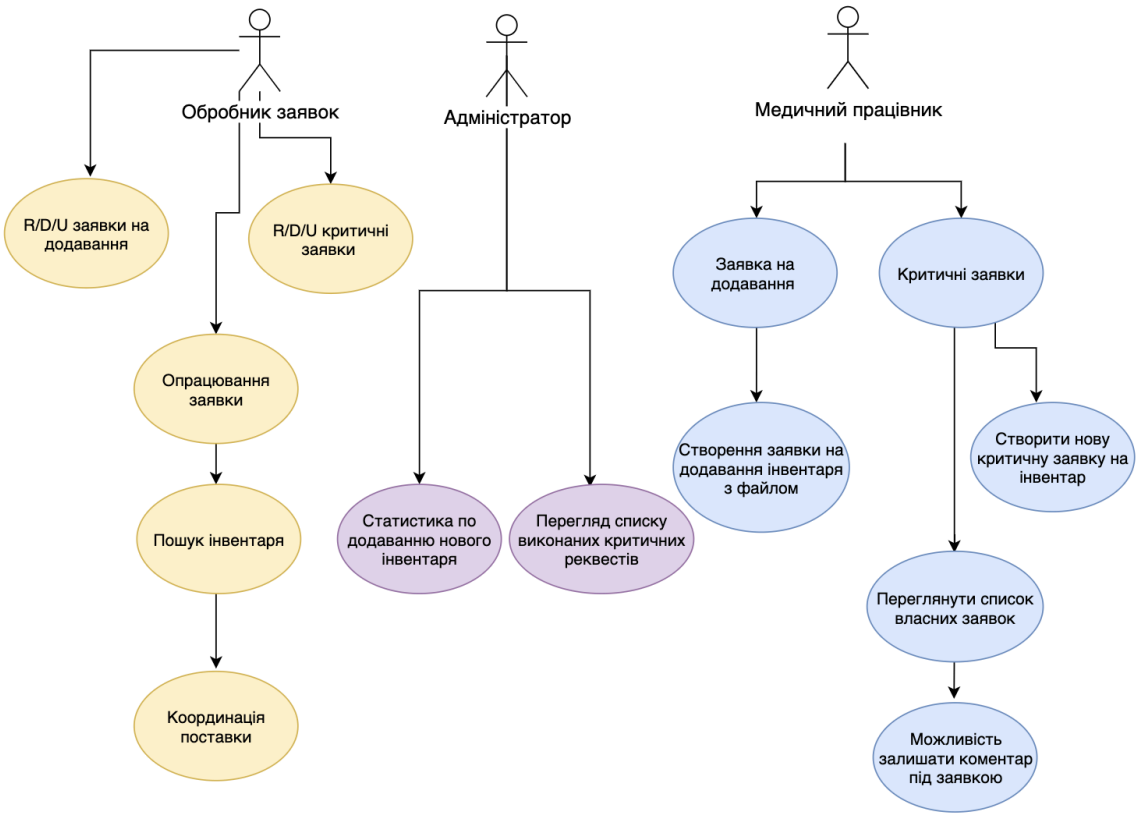


Рис. 3.1. Дерево функціональності різних користувачів системи

Розглянемо більш детально кожну з ролей:

- обробник заявок – співробітник міністерства охорони здоров’я, що використовує певну функціональність системи для обробки створених заявок, як і критичних так і заявок на додавання інвентаря;
- медичний працівник – рядовий співробітник лікарні, що може створювати заявки на додавання інвентарю до системи, чи заявку на кричну необхідність певного інвентаря в лікарні;
- адміністратор – людина, з вищими посадовими повноваженнями, що має доступ до загальної статистики.

Система містить два ключових процеси: створення та опрацювання критичної медичної заявки по необхідному інвентарю та створення заявки на додавання інвентарю до сховища даних. Кожен процес складається з окремих кроків. Весь процес називається життєвим циклом заявки. Початком життєвого циклу заявки вважається її створення, кінцем – підтвердження медичним працівником. На кожному з кроків обробник заявок може її скасувати або відправити на доопрацювання. Розглянемо спочатку процес створення критичної заявки, що наведено на рис. 3.2.

Рис. 3.2. Процес створення критичної заявки інвентаря

Спочатку, перевіряється наявність в лікарні з якої створюється заявка медичного обладнання, для якого створюється заявка, це автоматизована перевірка, для того, щоб зменшити навантаження на обробників заявок. Якщо такий інвентар існує, то користувач пере адресується на сторінку цього інвентаря. Якщо ж такого інвентаря немає – користувач заповнює форму та створює заявку. Далі до життєвого циклу заявки підключається обробник заявок, який перевіряє її на коректність та або відправляє на доопрацювання або починає опрацьовувати її. Обробник заявки певну має функціональність для пошуку схожого інвентаря в базі даних, після чого він переводить заявку в стан “В процесі”, та координує її поставку. В кінці медик, що створив заявку підтверджує її прибуття. Також обробник заявок та медик, можуть спілкуватися між собою шляхом коментування в конкретній заявці. В кінці адміністратор може побачити історію всієї заявки та отримати повну статистику по заявкам в системі.

Далі розглянемо процес створення заявки на додавання інвентарю до бази даних. Медичний працівник має створити фотографії певного інвентаря який він хоче додати до бази даних. Далі він заповнює форму, прикріплює фотографії та зберігає заявку. Далі в процес підключається обробник заявок, та оброблює заявку, перевіряє її на коректність. Якщо існують певні проблеми з заявкою, обробник відправляє її на доопрацювання, при цьому медик, що створював заявку отримує сповіщення про це. Якщо ж зауважень не виявлено, то новий інвентар потрапляє в базу даних. Медик, що створював заявку отримує сповіщення про це. Користувач з правами адміністратора, може також переглядати статистику створених заявок користувачами. Алгоритм процесу зображено на рис. 3.3.

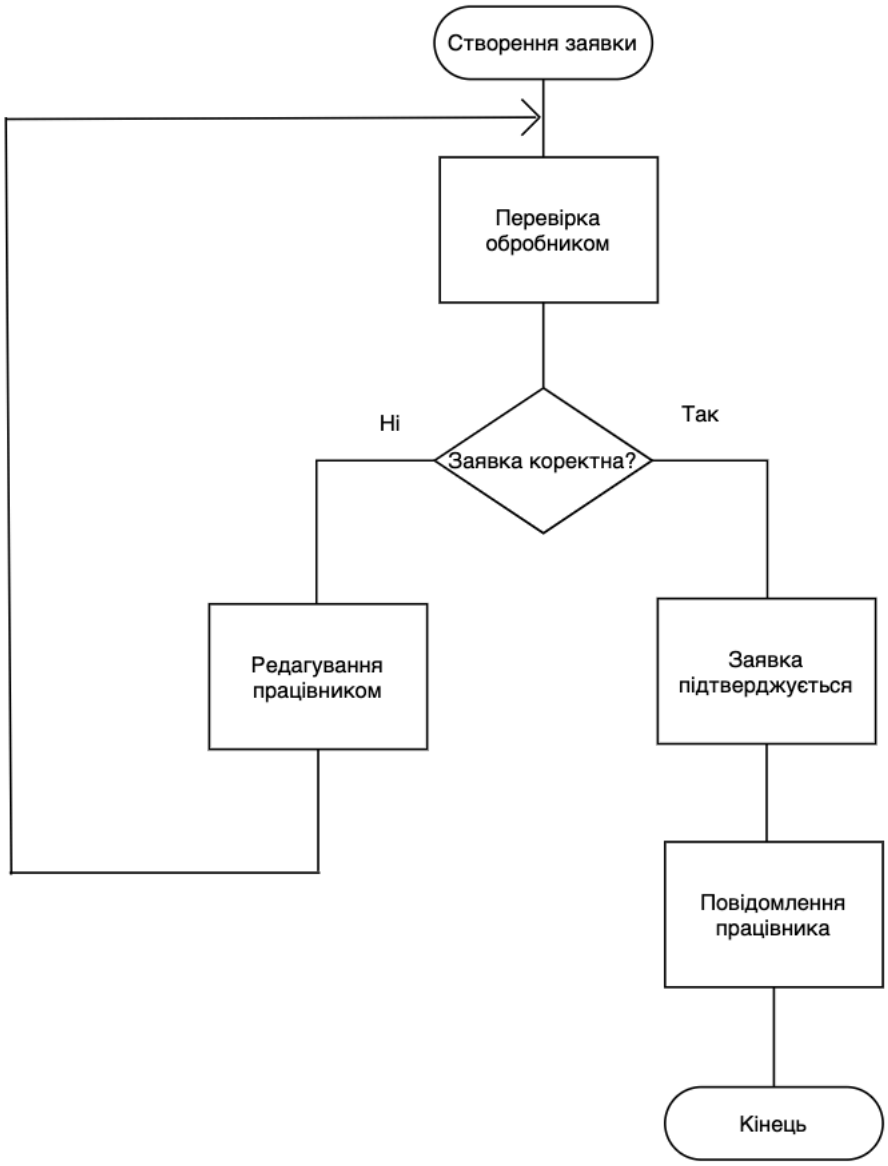


Рис. 3.3. Процес створення заявки на додавання інвентарю до бази даних

3.2. Функціональні вимоги до системи

Користувач при початку роботи в системі отримує логін та пароль від облікового запису, отже користувач має мати можливість авторизації у системі за рахунок логіну та паролю. Опишемо функціональні вимоги до серверної та клієнтської частини. Серверний застосунок має мати змогу:

- аутентифікувати користувача за допомогою логіна та паролю;
- авторизувати користувача, надати йому відповідні права до його ролі;

- сформувати JWT (Json Web Token) для подальшої роботи користувача з Web API;
- зберігати, видаляти та оновлювати нові критичні заявки на інвентар та заявки на додавання нового інвентарю;
- здійснювати повнотекстовий пошук серед інвентаря в базі даних по заданому тексту;
- фільтрувати заявки по заданим критеріям;
- відправляти сповіщення використовуючи для цього протокол обміну повідомленнями в реальному часі web sockets;
- агрегувати дані збережені в базі даних та формувати статистику.

В свою чергу клієнтська частина дипломної роботи має мати наступні функціональні можливості:

- мати систему навігації між сторінками веб-додатку;
- взаємодіяти з серверним застосунком за допомогою http запитів та технології AJAX;
- мати графічні компоненти інтерфейсу користувача для того, щоб ефективно використовувати серверні можливості для пошуку та фільтрації заявок.

3.3. Нефункціональні вимоги до системи

Система має відповідати наступним нефункціональним вимогам:

- мати зручний та адаптивний до різних видів пристроїв та розмірів екрану графічний інтерфейс користувача;
- система має мати безпечну систему авторизації та коректно працювати з даними користувача;
- дані користувача мають надійно зберігатися в базі даних, та у випадку втрати відновитися з резервних копій;
- користувачі мають отримувати зрозумілі та інформативні повідомлення про помилки в роботі системи.

3.4. Архітектура проєкту

3.4.1. Проєктування таблиць бази даних

Основні таблиці зі схеми бази даних:

- Request – містить інформацію про заявку, а саме: назву, дату створення, назву закладу, що створив заявку, тип заявки та статус, також історія коментарів, щодо заявки;
- Inventory – містить інформацію про інвентар, що був підтверджений обробником заявок. Обробник заявок проводить пошук інвентарю, саме в цій таблиці. Таблиця має наступні поля: назва інвентарю, дата створення, місце розташування, дані про габарити, кількість наявних одиниць, також дана таблиця має відношення one-to-many з таблицею прикріплених файлів;
- User – користувач системи, містить інформацію про користувача системи, та його роль: адміністратор, обробник заявки та медичний працівник. В залежності від ролі, користувачу надається доступ до тої чи іншої функціональності системи;
- InventoryRequest – таблиця в яку додається інвентар, який був створений в заявці на додавання інвентаря до бази даних, але ще не підтверджений обробником. У випадку, якщо обробник підтвердить коректність заявки, вона потрапить до таблиці Inventory.

Для роботи з базою даних було використано IDE DataGrip яка має можливості для візуального проєктування баз даних, моделювання та експлуатації. Кожна таблиця має внутрішній ключ, та зовнішні ключі для зв’язку з іншими таблицями. Таблиці нормалізовані до третьої нормальної форми. Також для збереження медіа файлів, було використано сторонній сервіс Minio, який дає змогу зберігати файли різноманітного формату, після успішного збереження, сервіс повертає посилання на збережений ресурс, яке можна зберегти до бази даних. За допомогою посилання можна отримати файл для завантаження та відображення.

3.4.2. Архітектура серверної частини

Розглянемо архітектуру серверної частини застосунку. Було вирішено реалізувати монолітний сервер, так як, монолітний застосунок потребує набагато меншої інфраструктурної складності в порівняні з мікро сервісним. На базі монолітного застосунку побудувати RESTful API та використати його для взаємодії з клієнтським застосунком. Також для генерації моделей відображення даних (data transfer objects) було використано бібліотеку swagger-codegen, яка відповідає специфікаціям OpenApi, вона надає змогу генерувати класи моделей відображення даних для клієнтського застосунку, що явно зменшує виконання рутинної роботи по опису даних, що надходять до клієнтського застосунку через web Api. За основу взято використання фреймворку Nestjs, для об’єктного відображення таблиць бази даних використано ORM TypeORM, який дає змогу описувати дані в таблицях у вигляді TypeScript класів. Також TypeORM має конструктор для виконання запитів до бази даної, або ж надає можливість використовувати звичайний SQL-синтаксис для цього. Додаток має шарову архітектуру, а саме для кожної дії використовується певний шар, а саме:

- доступ до даних – для доступу до даних та маніпуляцій з ними використовуються сервіси. Сервіси використовують з’єднання з базою даних, за допомогою впровадження залежностей TypeORM класу в клас сервісу;
- Data Transfer Objects – шаблон проєктування, що використовується для передачі даних;
- web API – для побудови RESTful сервісу використовуються контролери, які викликають методи певних сервісів для доступу до даних. Сервіси впроваджуються в конструктори класів контролерів за допомогою механізму впровадження залежностей. Сервіси використовують декоратори з бібліотеки swagger-codegen, що потім дає змогу згенерувати сервіси у клієнтському застосунку.

Nestjs дає змогу будувати модульний застосунок, що легко масштабується. Ключовим є використання механізму впровадження залежностей, це дає змогу дотримуватися основних концепцій об’єктно-орієнтованого проєктування. Також використання механізму впровадження залежностей дозволяє писати менш зв’язані між собою модулі, які нічого не знають про реалізацію інших модулів додатку. Це відповідає принципу інверсії залежностей, та принципу єдиного обов’язку. Всі класи, що можуть бути впроваджені в інші класи є провайдерами (Providers), вони використовують декоратор Injectable, що робить клас готовим до впровадження. Для модульності фреймворк використовує класи з декоратором Module, він надає метадані, які Nestjs використовує для організації структури додатку. Nest – застосунок має кореневий модуль, до якого підключаються всі інші модулі. Модулі за замовчуванням є сингелтонами, що дає змогу підключати один модуль до кількох різних модулів.

Для реалізації процесів ідентифікації, аутентифікації та авторизації користувача було вибрано технології JWT (json web tokens), для перевірки доступу Nest використовує guards механізми. Клас, що надалі буде слугувати для перевірки доступу до того чи іншого ресурсу має реалізувати інтерфейс CanActivate, після цього цей клас буде виступати декоратором певного методу контролера, та визначати чи достатньо прав має користувач для доступу до ресурсу.

3.4.3. Архітектура клієнтської частини

Клієнтська частина побудована за використання фреймворку Angular, для стилізації сторінок було вибрано препроцесор sass, а саме scss. Важливою перевагою Angular є його спорідненість з Nestjs, він використовує ті самі принципи побудови проєкту, та використовує для цього ті самі механізми (modules, providers, interceptors тощо). Також було використано бібліотеку NgRx для управління глобальним станом застосунку. Angular використовує

компонентний підхід до побудови клієнтських застосунків оснований на принципах об’єктно-орієнтованого програмування. Найбільш важливим в архітектурі клієнтських застосунків є визначити, як буде відбуватися наступних п’ять процеси:

1. Як потоки обміну даних будуть організовані всередині застосунку. Та як різні компоненти будуть обмінюватися даними.
2. Як реагувати та опрацьовувати взаємодію користувача з графічним інтерфейсом.
3. Як взаємодіяти з віддаленим RESTful API.
4. Як правильно працювати з станом даних, забезпечити узгодженість даних між собою.
5. Як організувати права доступу до того чи іншого ресурсу в залежності від прав користувача.

Опишемо окремо реалізацію кожного процесу в нашому клієнтському застосунку. Для реалізації обміну даних було використано механізм прив’язки даних між собою, цей механізм реалізовано у Angular за допомогою використання декораторів Input та Output, за рахунок яких можна організувати односторонній потік даних між компонентами. Було використано популярний шаблон для побудови клієнтських застосунків під назвою “Smart-dump components”, що полягає у розбиті компонентів на ті що мають певну логіку (smart components), та ті, що слугують лише компонентами для відображення (dump components) та отримують дані за рахунок прив’язки між компонентами.

Для опрацювання взаємодії користувача з графічним інтерфейсом Angular має механізм EventEmmiter`a, що надає реагувати та опрацьовувати події, що відбулися в dump components, та змінювати певні дані за рахунок прив’язки даних. Angular опирається на використання бібліотеки Rx.js, що дає змогу зробити компоненти реактивними, такими що реагують на зміну певних змінних моментально. Для того, щоб об’єкт був реактивним він має бути Observable, або ж BehaviorSubject. Observable являє собою масив подій,

що пов’язані з певним об’єктом. Rx.js надає широкий вибір для створення, модифікації observable потоків. Відмінність від Observable та BehaviorSubject полягає в тому, що значення BehaviorSubject об’єкта можна встановити за допомогою виклику методу, на відміну від Observable, який надає можливість лише модифікувати його значення, проте не встановити.

Для взаємодії з віддаленим RESTful API, в проєкті використовується згенерований за допомогою ng-swagger-gen сервіс на основі моделей та контролерів серверної частини додатку. Даний сервіс використовує HttpClient для взаємодії з віддаленим API за рахунок HTTP протоколу.

Для організації узгодженості даних вирішено використовувати бібліотеку NgRx, вона є одною з реалізацій шаблону проєктування flux, та використовує бібліотеку Rx.js. NgRx допомагає організувати дерево глобального стану застосунку, до якого мають доступ всі компоненти. Весь стан додатку поділений на локальний та глобальний. Локальний стан – це стан конкретного компонента, про який не мають знати інші компоненти, він є інкапсульованим. Глобальний стан – стан, що є доступний всім компонентам додатку, глобальний стан можна використовувати для комунікації компонентів, які не зв’язані між собою через прив’язку даних. Підхід, що використовує NgRx полягає у використані наступних речей:

- Actions – події, звичайна функція, яка є сигналом для виконання певної операції;
- Reducer – редюсери, функція, що обробляє нові дані, та змінює значення стейту. Редюсери є іммутабельними чистими функціями, це означає, що значення таких функцій повино залежати, лише від вхідних параметрів та повертати новий екземпляр стану;
- Effects – ефекти, так як редюсери, мають бути чистими функіями, потрібно мати місце в якому мають виконуватися сайд-ефекти, операції значення яких не можна обрахувати на основі вхідних даних. Прикладом сайд-ефектів можуть слугувати запити до Web

API, виклик графічних повідомлення для інтерфейсу користувача. Ефекти, реагують на виклик певної події, виконують певні запити, та повертають іншу подію, як результат свого виконання;

- Selectors – селектори, функції, які використовуються для доступу до дерева стану. Селектори повертають значення необхідного вузла дерева. Вони є мемоїзованими, такими, що кешують значення всіх викликів, і при наступному такому ж виклику обраховують його на основі попередніх значень.

Життєвий цикл роботи бібліотеки NgRx наведено на рис. 3.4.

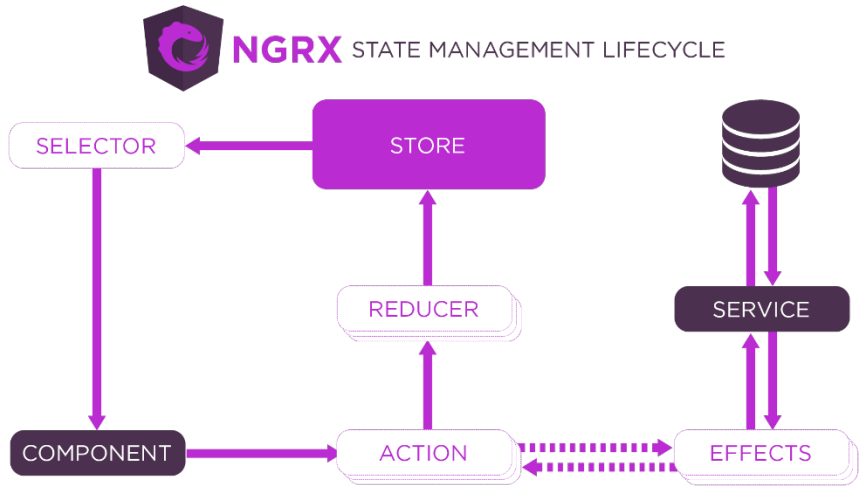


Рис. 3.4. Життєвий цикл роботи бібліотеки NgRx [21]

Заключним є питання вирішення визначення прав доступу до певних ресурсів у відповідності до прав користувача. Для цього було використано бібліотеку з відкритим кодом ngx-permissions, вона дає змогу обмежувати доступ до певних компонентів за рахунок структурних директив у html-розмітці на основі прав доступу користувача.

3.5. Висновки

У даному розділі було проаналізовано реалізацію серверної та клієнтської частини додатку, а також організацію бази даних. В наслідок використання споріднених технологій, таких як Nestjs та Angular та

використання Swagger для генерації моделей даних та сервісів було розроблено надійну архітектуру додатку, що може легко змінюватись та розширятись у разі необхідності розвитку застосунку. Високорівневу архітектуру додатку зображено на рис. 3.5. За рахунок використання механізму впровадження залежностей розроблена архітектура дає можливість розроблювати бізнес-логіку без прив'язки до конкретної реалізації. Також за рахунок використання інверсії залежностей можливо краще здійснювати тестування додатку, заміняючи реальні сервіси тестовими. Для завантаження файлів використовується сторонній сервіс Minio, що дає змогу зберігати файли будь-якого формату. Даний піхід використовується задля зменшення об'єму бази даних та підвищення її швидкодії.

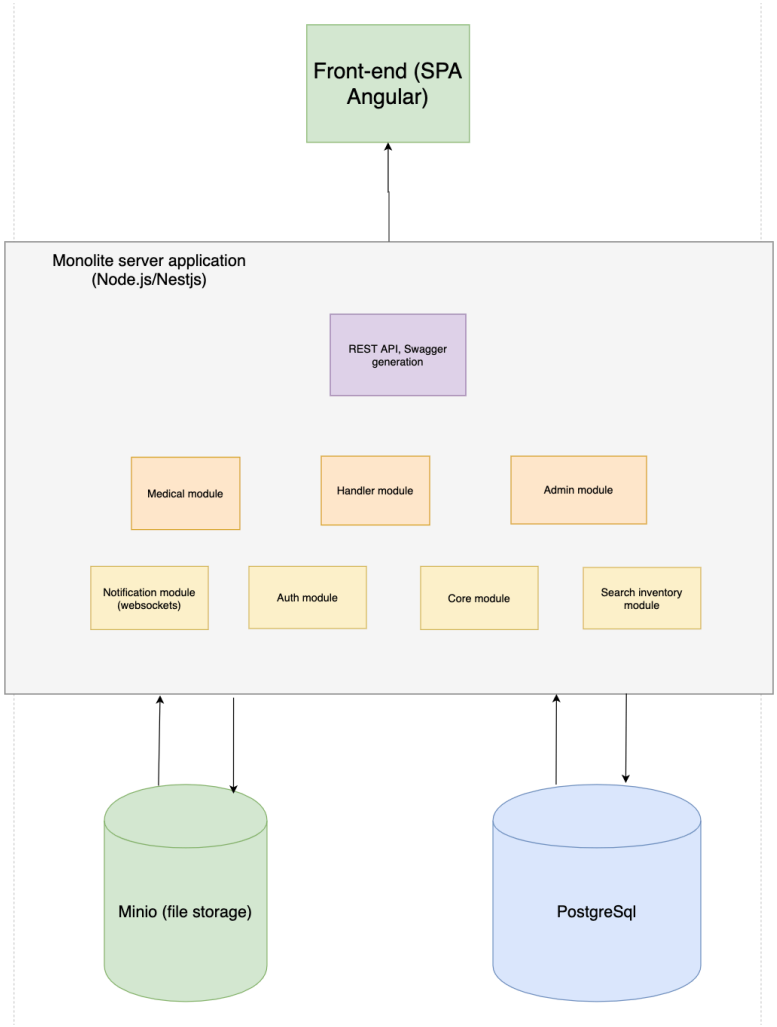


Рис. 3.5. Діаграма високорівневої архітектури додатку

4. АНАЛІЗ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Основні алгоритми використання розробленого програмного забезпечення

4.1.1. Процес створення критичної заявки для інвентаря

Після успішної авторизації, користувач може створити заявку для критично необхідного інвентаря. Для цього він має перейти на сторінку критичних запитів, використавши для цього навігацію сайту, що наведено на рис. 4.1.

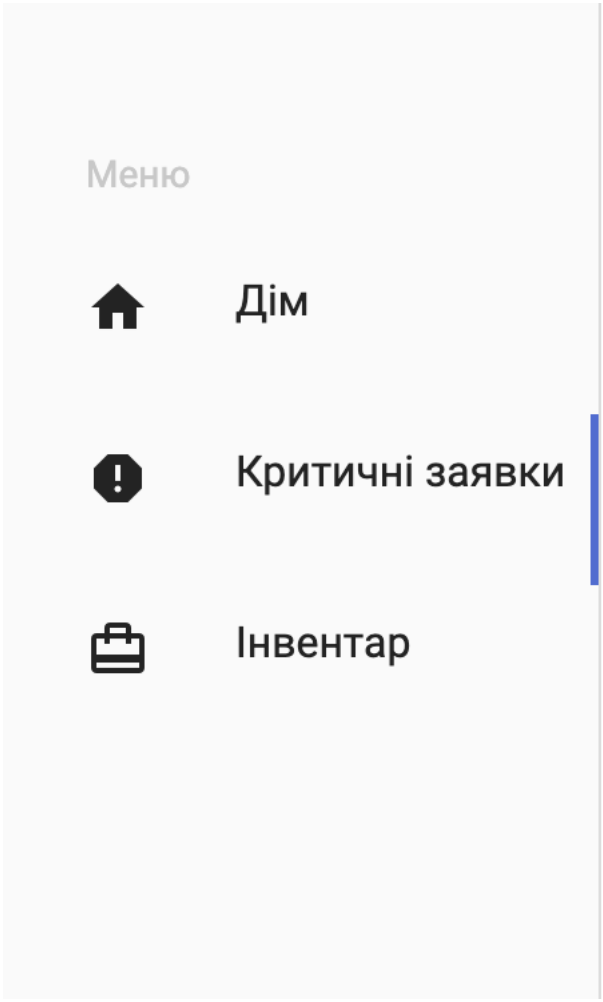


Рис. 4.1. Область навігації сайту

Синій індикатор є маркером вибраної поточної сторінки. Після переходу на сторінку критичних заявок, користувач бачить дві вкладки: «Створити заявку» та «Переглянути створені» заявки. На вкладці Створити

заявку, користувач може заповнити форму для створення заявки. Вказавши там необхідні дані: назву інвентарю, номер лікарні, виробник інвентаря, адреса лікарні та опис. Інтерфейс для створення заявки для критично необхідного інвентаря наведено на рис. 4.2.

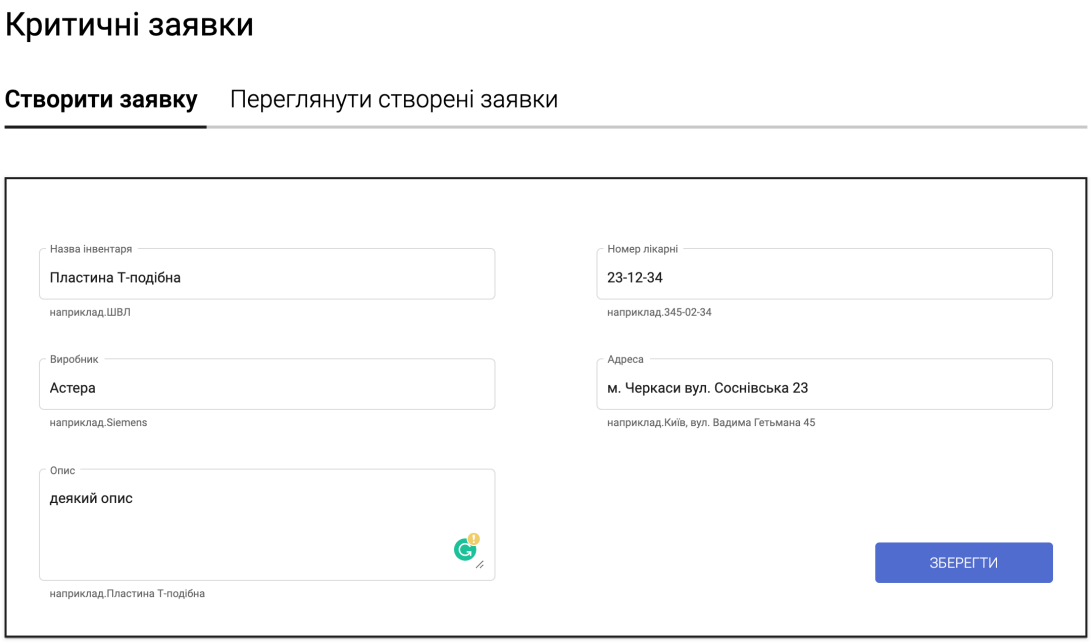


Рис. 4.2. Інтерфейс для створення заявки для критично необхідного інвентаря

Кожне поле має коротку підказку, приклад даних, що треба записати до поля. Після того як користувач ввів всі необхідні поля, він зберігає заявку та відправляє її на подальше опрацювання. Інтерфейс для перегляду створених критичних заявок наведено на рис. 4.3.

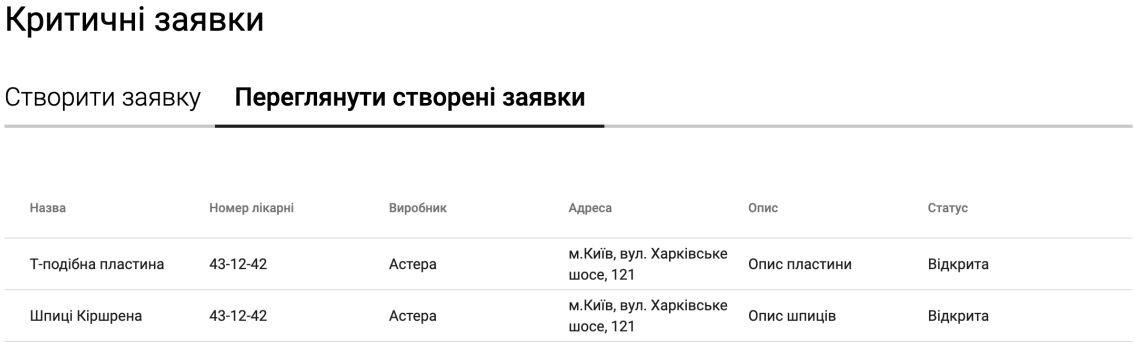


Рис. 4.3. Інтерфейс для перегляду створених критичних заявок

Вибравши вкладку «Переглянути створені заявки» користувач може переглянути статус заявки. Заявка може перебувати в наступних чотирьох станах:

- відкрита – заявка після створення користувачем, набуває статусу відкритої це дозволяє в подальшому обробникам заявок шукати та сортувати заявки за статусом;
- потребує доопрацювання – якщо обробник заявки, виявляє певні помилки та заявка потребує певного доопрацювання та уточнення;
- підтверджена – якщо в заявці немає помилок, всі дані зазначено вірно, обробник заявки переводить її в статус підтверджена. Це сигналізує користувачу, що створив заявку, про те, що почався процес доставки інвентаря в зазначену лікарню;
- закрита – після виконання всіх операцій заявка набуває статусу закритої, та потім фігурує лише в статистичних даних.

4.1.2. Процес створення заявки на додавання інвентаря до бази даних

Для створення заявки на додавання користувач заповнює форму, яка схожа на форму створення критичної заявки, в формі він вказує необхідні дані та прикріпляє файли, що стосуються інвентаря який він хоче додати до бази даних, це можуть бути як і медіа файли, так і текстові документи. Серед необхідних до заповнення полей форми необхідно заповнити наступні: назва інвентаря, дата виробництва, галузь застосування, виробник, адреса лікарні в якій знаходиться даний інвентар, кількість в наявності. Після заповнення всіх полей користувач зберігає заявку, бачить попередній вигляд заявки та відправляє її на опрацювання.

4.1.3. Процес пошуку інвентаря оброблювачем заявок

Оброблювач заявки має функціональність для пошуку інвентаря за певним списком параметрів. Інтерфейс для пошуку інвентаря наведено на рис. 4.4.

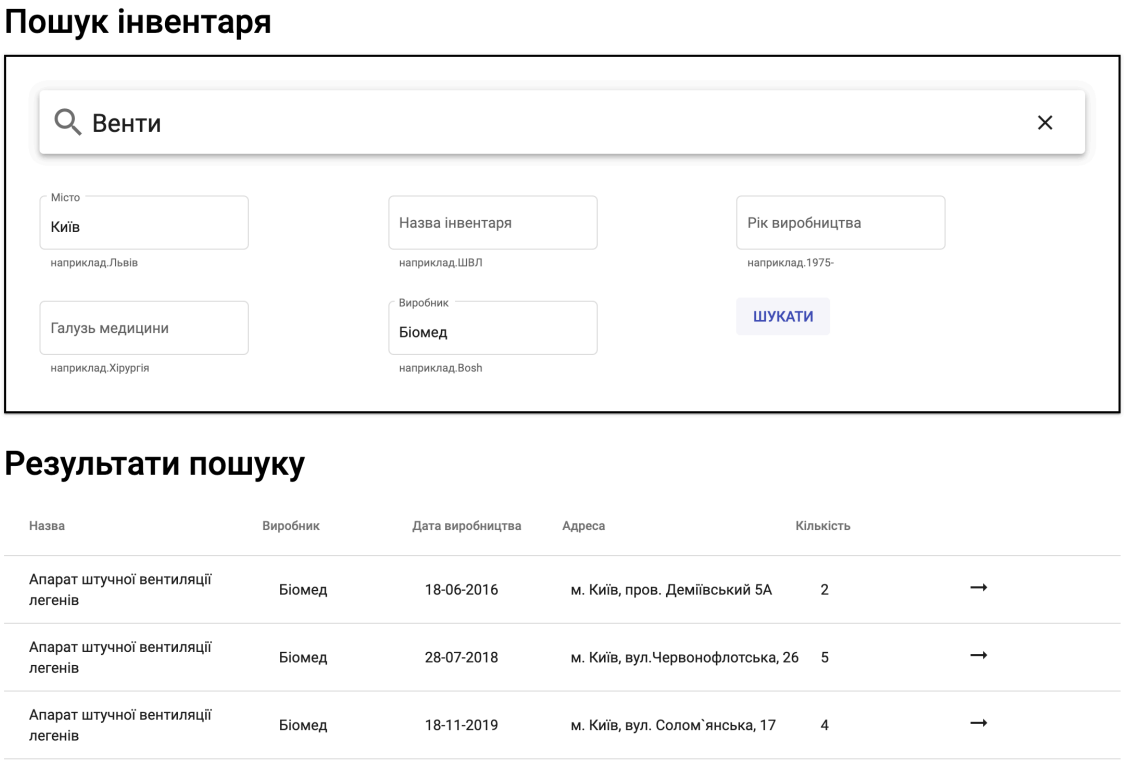


Рис. 4.4. Інтерфейс для пошуку інвентаря та результати пошуку

Для введення параметрів пошуку використовується форма пошуку, яка складається з:

- текстового поля для введення ключових слів, за якими можливо знайти інвентар. Далі це значення буде використовуватися за повнотекстового пошуку серед записів базі даних;
- місто – текстова поле, назва міста знаходження інвентаря;
- назва інвентаря – текстова поле, назва інвентаря, можливо не повна;
- рік виробництва – конкретній рік, або ж проміж з якого вироблявся певний інвентар;

- галузь медицини – галузь в якій застосовується даний інвентар;
- виробник – компанія яка є виробником даного інвентаря.

Користувач може зазначити не всі поля, проте кожне додане значення підвищить точність результатів пошуку. Після того як користувач натиснув на кнопку «Шукати», відбувається пошук та з’являються результати пошуку. Обробник заявок може вибрати конкретний інвентар та перейти на його сторінку.

4.1.4. Агрегація статистичних даних для користувачів адміністраторів

Користувачі системи з правами адміністратора мають доступ до перегляду статистичної інформації, щодо створених заявок з критично необхідного інвентаря, та заявок на додавання інвентаря до бази даних.

На рис. 4.5 зображено кругову діаграму статусів критичних заявок в базі даних. На діаграмі адміністратор бачить скільки заявок з відповідними статусами наразі в системі.

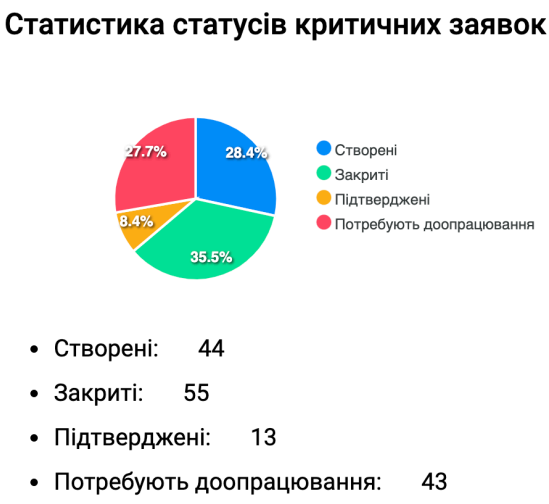


Рис. 4.5. Статистичний блок статусів критичних заявок

На рис. 4.6 показано лінійну діаграму кількості створених заявок у відповідності до місяця. Користувач може вибрати точку на лінію графіку та побачити число створених критичних заявок цього місяця.

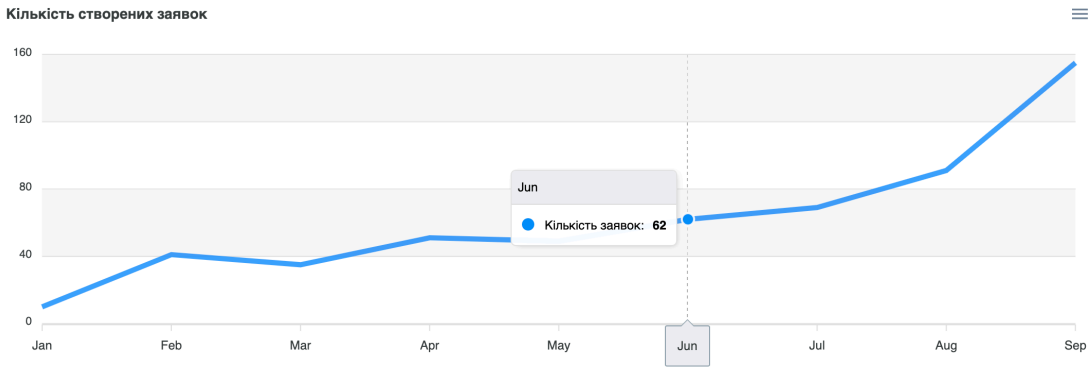


Рис. 4.6. Лінійна діаграма, кількості створених заявок

На рис. 4.7 показано діаграму співвідношення створених користувачами заявок на додавання інвентаря до бази даних до заявок які були підтвердженими обробниками заявок та доданими.

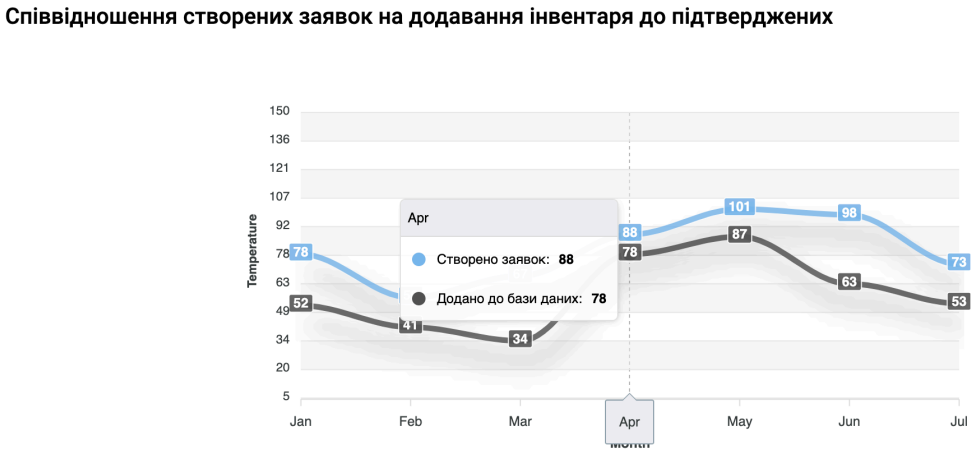


Рис. 4.7. Діаграма співвідношення створених заявок до підтверджених

4.2. Тестування веб-додатку

Тестування програми є важливою та невід'ємною частиною процесу створення програмного забезпечення. Цей процес спрямований на знаходження помилок в розробленому програмному забезпечені, відповідності розробленого ПЗ до вимог та відслідковування коректності роботи програми. Основними видами тестування є три підходи: юніт-тестування, інтеграційне тестування та наскрізне тестування. Кожен з підходів використовується для тестування певної частини програми. Юніт-тести використовуються для того, щоб протестувати найменші частини програми: функції, логічні умови тощо, відповідальність, щодо юніт-тестування лежить на розробниках. Інтеграційне тестування використовується для тестування взаємодії декількох програмних модулів. Наскрізне тестування використовується для цілісної перевірки роботи всієї програми. Тестування також поділяють на автоматизоване та ручне, в дипломній роботі було застосовано ручне тестування, так як розмір програми не великий.

В ході виконання дипломного проєкту та розроблення веб-додатку, тестування виконувалось після кожного етапу написання коду для певних частин функціональності та загалом для всієї програми.

В процесі тестування можна виділити основні етапи:

- ініціювання (вибір необхідних модулів для тестування, браузерів та пристроїв);
- планування (вивчення методів та підходів, щодо тестування, що використовувалися в аналогічному програмному забезпечені);
- тест – дизайн (створення тестових випадків та тестових сценаріїв, в англомовній літературі – test cases);
- виконання тестів (безпосереднє тестування вибраної функціональності);
- аналіз та звітність (збирання інформації про проведені тести, оновлення статутів тестових сценаріїв);

- завершення (рекомендації щодо подальших дій по усуненню помилок та вдосконаленню програми).

Етапи планування, написання тестових випадків можуть повторюватися безліч кількість раз поки програмне забезпечення не почне відповідати всім вимогам.

Процес тестування дипломної роботи відбувався наступним чином: спочатку виконувалися юніт-тести, це дає змогу впевнитися, що окремі найпростіші складові працюють коректно, і далі можна приступати до перевірки тестових сценаріїв. Для тестування було використано певні утиліти, наприклад Postman, що дає змогу здійснювати запити до HTTP сервера та бачити їх результати, за допомогою цього було про тестовано розроблений Web API. Для виконання тестів не обов'язково використовувати реальну базу даних, для цього створюються фальшиві дані, які симулюють роботу бази даних. Інтеграційне тестування було проведено методом чорної скриньки, (англ. «black-box testing») – це тестування програмного забезпечення з точки зору зовнішнього світу, що не використовує знання про внутрішнє влаштування програмного забезпечення.

Розглянемо тестові сценарії, що були створені відповідно до вимог тестування в аналогічних розробках і були використані для перевірки коректності роботи програми наведені у таблиці 1.

Таблиця 1

Тестові сценарії

№	Функція	Дії	Очікуваний результат
1	2	3	4
1	Авторизація	Заповнити всі обов'язкові поля коректними даними	Успішна авторизація

Продовження табл. 1

2	Авторизація	Ввести логін та пароль, що не відповідає даному логіну	Повідомлення про помилку
3	Авторизація	Не заповнити обов’язкове поле	Повідомлення про помилку
4	Збереження створеної заявки	Користувач надсилає коректно заповнені дані	Нова заявка успішно створена
5	Збереження створеної заявки	Користувач заповнює певні поля некоректно	Повідомлення про некоректно заповнені дані у графічному інтерфейсі користувача
6	Запит на отримання списку створених заявок	Користувач не має необхідних прав на доступ до даної інформації	Повідомлення про відсутність прав доступу до даного ресурсу
7	Пошук інвентаря	Обробник заявок коректно заповнює всі поля форми та надсилає запи	Знизу під формою пошуку з’являється таблиця результатів пошуку
8	Пошук інвентаря	Користувач вводить не валідні символи в строку пошуку	Повідомлення про некоректно заповнені дані у графічному інтерфейсі користувача
9	Пошук інвентаря	Користувач надсилає коректні дані, проте за даним запитом інвентаря немає	Повідомлення про те, що запит відбувся успішно, проте результатів не знайдено

На етапі ініціації тестування було вирішено протестувати працездатність та зовнішній вигляд веб-додатку в різних браузерах, таких як Mozilla Firefox, Google Chrome та Safari.

За допомогою тестування були отримані результати щодо вдосконалення та доопрацювання програми веб-додатку.

4.3. Рекомендації щодо подальшого вдосконалення

Розроблений веб-додаток є концептом системи для створення заявок з критично необхідного медичного забезпечення. Сервіс може використовуватися у реальному житті, тому що містить достатню кількість функціональних можливостей для повноцінної роботи. Він містить мінімальний необхідний функціонал, що потрібний для життєвого циклу заявки. Розроблена архітектура дозволяє легко додавати новий функціонал та розширювати можливості додатку.

Серед можливих покращень можна виділити наступні:

- чат в реальному часі між користувачем, що створив заявку та обробником заявки, що створюється відносно створеної заявки;
- графічне відображення історичної хронології операцій над заявкою;
- більш деталізована статистика щодо створених заявок та доданого інвентаря.

ВИСНОВКИ

Метою даного дипломного проєкту було створення веб-додатку для створення, аналізу та опрацювання заявок з критично необхідного медичного інвентаря.

Після проведення аналізу існуючих аналогів, було прийняте рішення, що система буде складатися з реляційної бази даних, монолітного серверу та клієнтського веб застосунку.

На основі проведеного аналізу щодо вибору засобів реалізації, було вирішено створити серверну та клієнтську частину на мові програмування TypeScript. Для реалізації серверної частини було використано фреймворк Nestjs, для клієнтської частини – Angular. Як базу даних було вибрано СКДБ PostgreSQL. Також було використано хмарний сервіс для збереження медіа файлів Minio.

Розроблений додаток:

- надає можливість авторизуватися та отримувати доступ до ресурсів додатку, які відповідають ролі користувача;
- надає можливість створювати заявки з критично необхідного медичного інвентаря;
- дозволяє обробляти створені заявки;
- має графічний інтерфейс для роботи з робленою функціональністю та є доступним для різних браузерів та пристроїв
- має публічний API для роботи з створеними заявками.

Розробка програми виконана у повному обсязі та відповідає поставленим вимогам.

Використання розробленої системи дозволить автоматизувати частину медичних процесів та значно підвищити їх ефективність.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

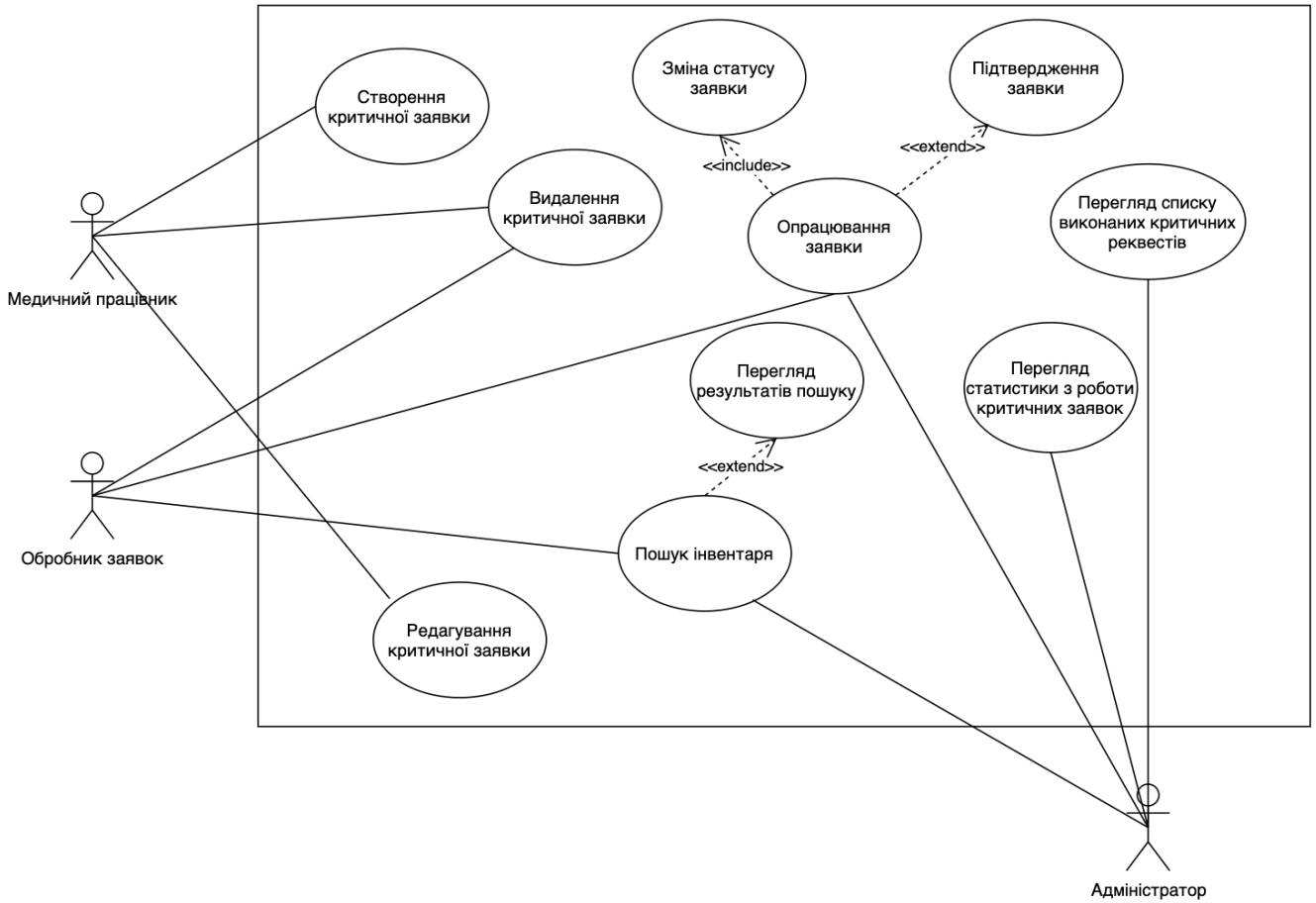
1. Проксі-сервер : [Електронний ресурс]. Режим доступу: https://cutt.ly/8yNG7nl
2. Що таке хмарні технології та як це працює? : [Електронний ресурс]. Режим доступу: https://futurenow.com.ua/shho-take-hmarni-tehnologiyi-ta-yak-tse-pratsyuye/
3. Порівняння SQL та NoSQL : [Електронний ресурс]. Режим доступу: https://habr.com/ru/company/ruvds/blog/324936/
4. A Look at 5 NoSQL Solutions : [Електронний ресурс]. Режим доступу: https://dzone.com/articles/a-look-at-5-nosql-solutions
5. Багатоплатформність : [Електронний ресурс]. Режим доступу: https://cutt.ly/KyNHnSq
6. SOLID принципи: Пояснення та приклади : [Електронний ресурс]. Режим доступу: https://itnext.io/solid-principles-explanation-and-examples-715b975dcad4
7. Java Spring пакети : [Електронний ресурс]. Режим доступу: https://docs.spring.io/spring/docs/2.5.6/javadoc-api/org/springframework/core/package-use.html
8. Best Python libraries for Machine Learning : [Електронний ресурс]. Режим доступу: https://geeksforgeeks.org/best-python-libraries-for-machine-learning/
9. Система типізації : [Електронний ресурс]. Режим доступу: https://cutt.ly/5yNHHhk
10. Машиний код : [Електронний ресурс]. Режим доступу: https://cutt.ly/qyNHXdy
11. TypeScript за 30 хвилин. : [Електронний ресурс]. Режим доступу: https://codeguida.com/post/475
12. Language Server Protocol : [Електронний ресурс]. Режим доступу: https://cutt.ly/qyNJuAZ

13. Реактивне програмування : [Електронний ресурс]. Режим доступу: https://cutt.ly/9yNJJp4
14. Основні парадигми програмування : [Електронний ресурс]. Режим доступу: https://studfile.net/preview/5994723/
15. Віртуальний DOM і деталі його реалізації в React : [Електронний ресурс]. Режим доступу: https://ru.reactjs.org/docs/faq-internals.html
16. Big data : [Електронний ресурс]. Режим доступу: https://cutt.ly/5yNJ2Xf
17. Індекс таблиці бази даних : [Електронний ресурс]. Режим доступу: https://cutt.ly/0yNJ4EW
18. Рівні ізоляції транзакцій : [Електронний ресурс]. Режим доступу: https://habr.com/ru/post/469415/
19. SDK : [Електронний ресурс]. Режим доступу: https://cutt.ly/YyNKeOt
20. Транзакції (бази даних) : [Електронний ресурс]. Режим доступу: https://cutt.ly/vyNKn43
21. Документація NgRx : [Електронний ресурс]. Режим доступу: https://ngrx.io/guide/store

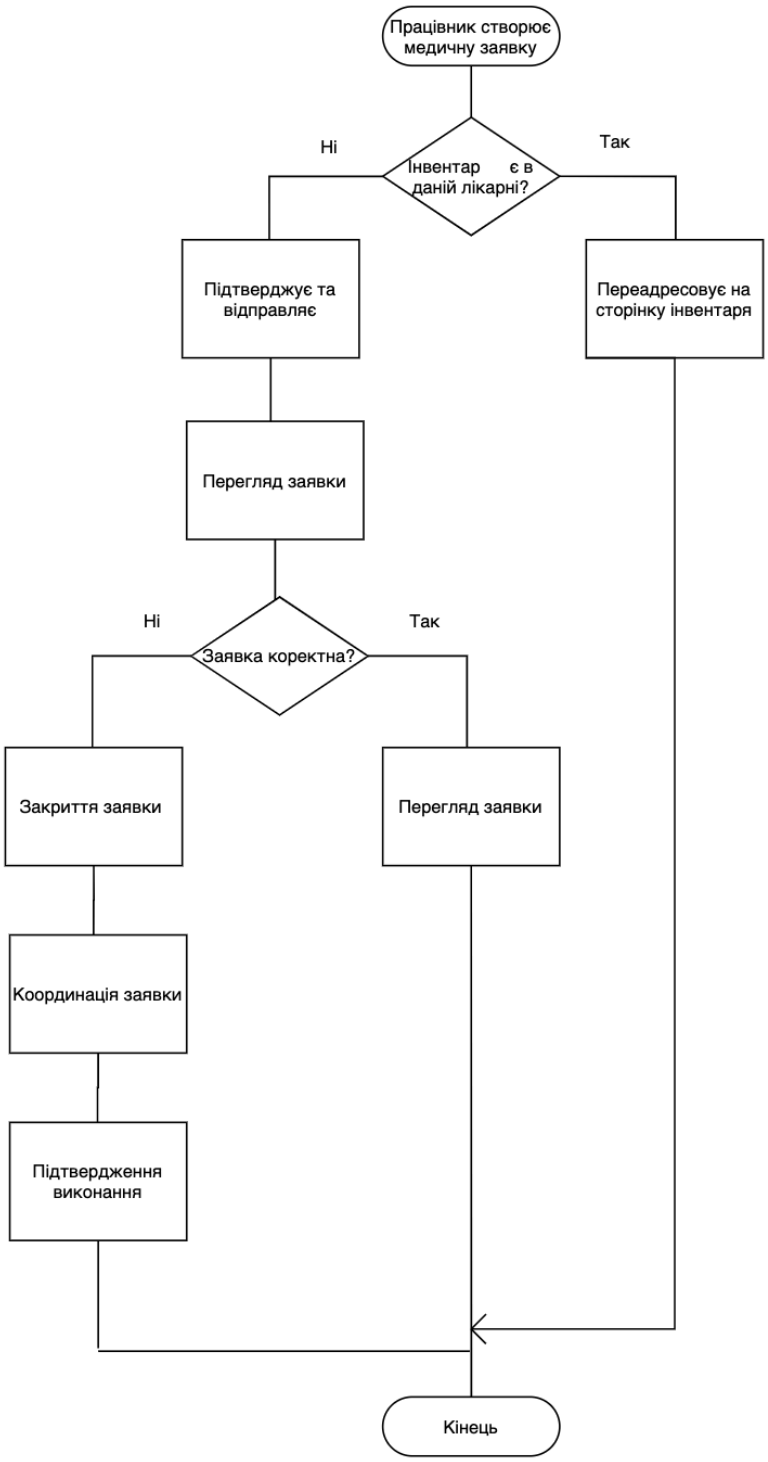
ДОДАТКИ

Додаток 1

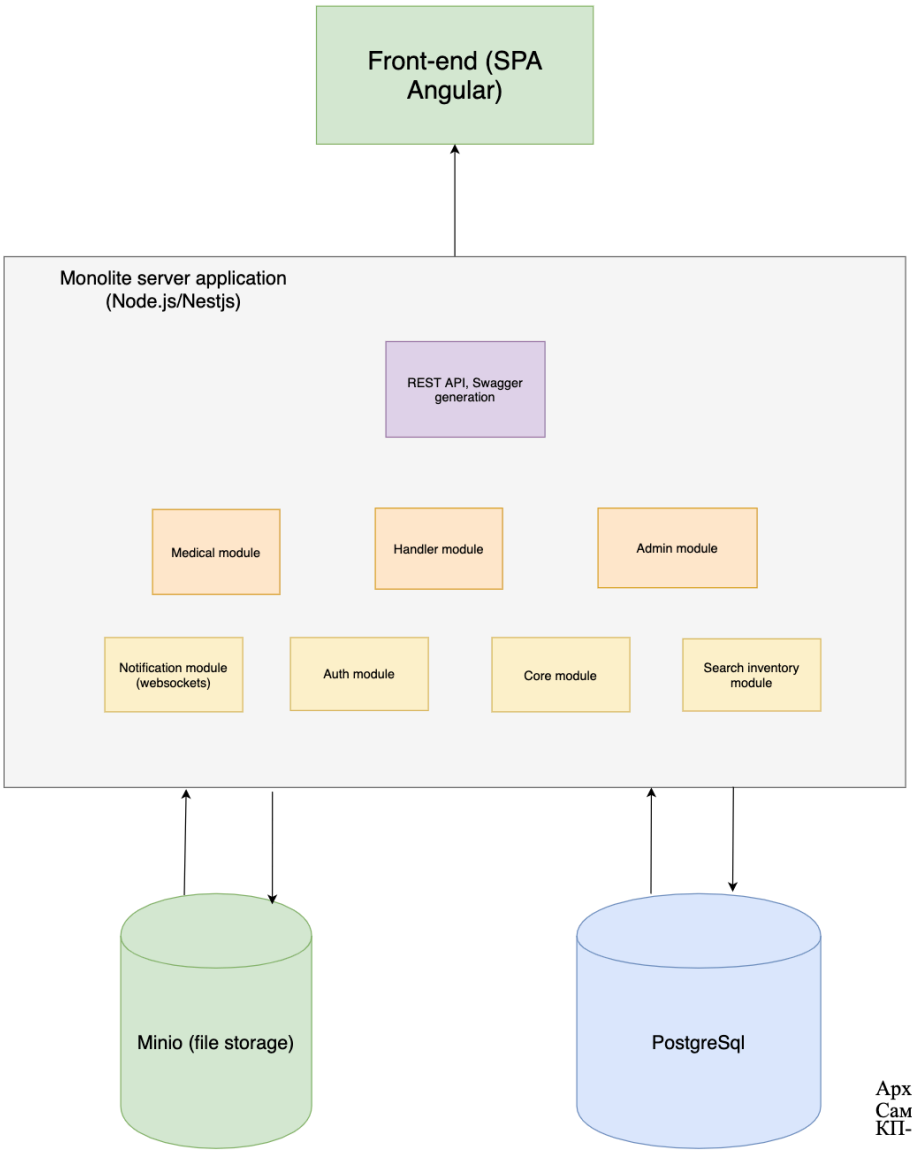
Копії графічних матеріалів



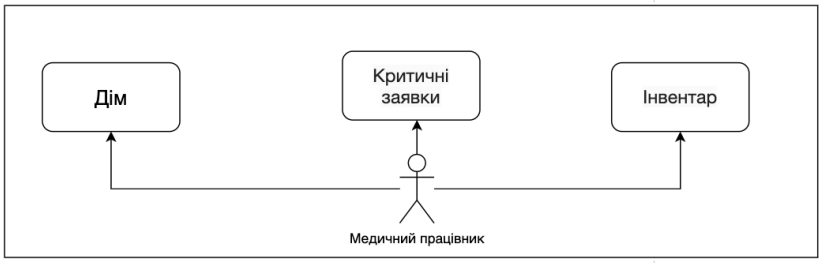
ДП.045440-06-99.
Аналітично-інформаційна система для створення критичних медичних заявок.
Діаграма розподілу прав доступу користувачів.
Діаграма прецедентів.

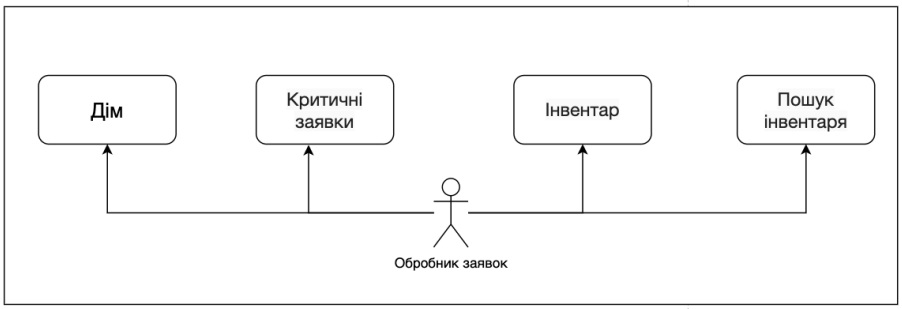


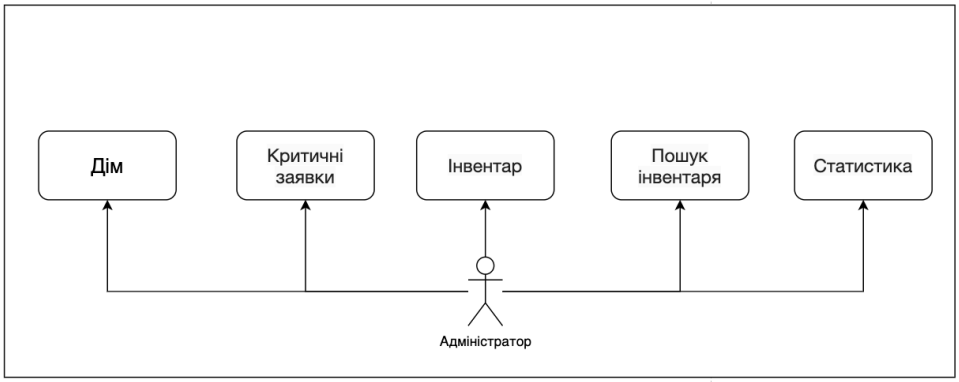
ДП.045440-06-99.
Аналітично-інформаційна система для створення критичних медичних заявок. Життєвий цикл заявки з критично необхідного інвентаря. Схема алгоритму



Архітектура веб-додатку
Самойленко Н.Ю., група КП-62







Структура веб - сторінок
Самойленко Н. Ю., група КП-62

Додаток 2

Лістинг програми

2.1. КОМПОНЕНТ ДЛЯ ПОШУКУ ІНВЕНТАРЯ

```
@Component({
  selector: 'app-inventory-search',
  templateUrl: './inventory-search.component.html',
  styleUrls: ['./inventory-search.component.scss']
})
export class InventorySearchComponent implements OnInit {
  displayedColumns: string[] = [
    'Name',
    'Manufacturer',
    'Date of release',
    'Location',
    'Count',
    'More'
  ];
  dataSource: MatTableDataSource<Inventory>;

  constructor() { }

  ngOnInit(): void {
    this.dataSource = new MatTableDataSource(inventories);
  }

  updateSearchTerm(term) {}

}
```

2.2. КОМПОНЕНТ НАВІГАЦІЙНОЇ МОДЕЛІ

```
@Component({
  selector: 'app-sidenavigation',
  templateUrl: './sidenavigation.component.html',
  styleUrls: ['./sidenavigation.component.scss']
})
export class SidenavigationComponent implements OnInit, OnDestroy
{
  readonly sideNavChoice = SideNavChoice;
  sidenavItems: Array<SideNavChoiceType> = [
    {
      name: 'Дім',
      icon: 'home',
      redirectTo: '',
      choiceType: SideNavChoice.Home
```

```
    },
    {
      name: 'Критичні заявки',
      icon: 'report',
      redirectTo: 'critical-request',
      choiceType: SideNavChoice.CriticalRequest
    },
    {
      name: 'Інвентар',
      icon: 'card_travel',
      redirectTo: 'inventory-request',
      choiceType: SideNavChoice.InventoryRequest
    },
    {
      name: 'Пошук інвентаря',
      icon: 'search',
      redirectTo: 'inventory-search',
      choiceType: SideNavChoice.InventorySearch
    },
    {
      name: 'Статистика',
      icon: 'list',
      redirectTo: 'statistic',
      choiceType: SideNavChoice.Statistic
    }
  ];
  selected: SideNavChoice;
  destroy$: Subject<boolean>;
  urlParam$: Observable<string> = this.store.select(selectUrl);

  constructor(private router: Router, private store: Store<State>)
{}

  ngOnInit(): void {
    this.urlParam$
      .pipe(
        map((url) => url.split('/')),
        switchMap((params) => this.getCurrentPage$(params)),
        // takeUntil(this.destroy$)
      )
      .subscribe((selected) => (this.selected = selected));
  }

  ngOnDestroy() {
    this.destroy$.next();
  }
```

```
  getCurrentPage$(urlParams: string[]): Observable<SideNavChoice>
{
    const current = this.sidenavItems.find((item) =>
      urlParams.includes(item.redirectTo)
    );
    const    page    =    current    ?    current.choiceType    :
SideNavChoice.Home;
    return of(page);
  }

  redirectToPage(url: string) {
    this.router.navigateByUrl(url);
  }
}
```

2.3. КОМПОНЕНТ ДЛЯ ВВЕДЕННЯ ТЕКСТУ ДЛЯ ПОШУКУ

```
@Component({
  selector: 'app-search-bar',
  templateUrl: './search-bar.component.html',
  styleUrls: ['./search-bar.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class SearchBarComponent implements OnDestroy, OnInit {
  searchForm: FormGroup;
  alive$: Subject<boolean>;
  @Output()   updateSearchTerm:   EventEmitter<string>   =   new
EventEmitter();
  @Input() placeholder: string;
  searchTerm = '';

  constructor(private fb: FormBuilder) {
    this.searchForm = this.fb.group({
      term: new FormControl('')
    });
  }

  initForm() {}

  ngOnInit() {
    this.searchForm
      .get('term')
      .valueChanges.pipe(debounceTime(200),
takeUntil(this.alive$))
      .subscribe((term) => {
```

```
        this.searchTerm = term;
        this.updateSearchTerm.emit(term);
      });
  }

  ngOnDestroy() {
    this.alive$.next();
  }

  clearSearchTerm() {
    this.searchForm.get('term').setValue('');
  }
}
```

2.4 HTML РОЗМІТКА БОКОВОЇ НАВІГАЦІЇ САЙТУ

```
<div class="sidenavigation">
  <div    fxLayout="column"    fxLayoutAlign="center    center"
class="items-box">
    <div class="main-menu-text">Меню</div>
    <div
      fxLayout="row"
      fxLayoutAlign="start center"
      fxLayoutGap="1rem"
      *ngFor="let item of sidenavItems"
      [ngClass]="sideNavChoice.CriticalRequest                ===
item.choiceType ? 'item selected' : 'item'"
      (click)="redirectToPage(item.redirectTo)"
    >
      <div
        fxLayout="row"
        fxLayoutAlign="space-between start"
        fxLayoutGap="2rem"
      >
        <div>
          <mat-icon class="icon">{{ item.icon }}</mat-icon>
        </div>
        <div>{{ item.name }}</div>
      </div>
    </div>
  </div>
</div>
```

2.5 HTML РОЗМІТКА КОМПОНЕНТУ ПОШУКУ ІНВЕНТАРЯ.

```
<div>
  <h1 class="section-name">Пошук інвентаря</h1>
  <div>
    <mat-card>
      <app-search-bar
        [placeholder]="'Введіть ключові слова'"
        (updateSearchTerm)="updateSearchTerm($event)"
      ></app-search-bar>
      <form [ngStyle]="{ 'margin-top': '2rem' }">
        <div class="search-form">
          <div>
            <mat-form-field appearance="outline">
              <mat-label>Місто</mat-label>
              <input matInput />
              <mat-hint>наприклад.Львів</mat-hint>
            </mat-form-field>
          </div>
          <div>
            <mat-form-field appearance="outline">
              <mat-label>Назва інвентаря</mat-label>
              <input matInput />
              <mat-hint>наприклад.ШВЛ</mat-hint>
            </mat-form-field>
          </div>
          <div>
            <mat-form-field appearance="outline">
              <mat-label>Рік виробництва</mat-label>
              <input matInput />
              <mat-hint>наприклад.1975-</mat-hint>
            </mat-form-field>
          </div>
          <div>
            <mat-form-field appearance="outline">
              <mat-label>Галузь медицини</mat-label>
              <input matInput />
              <mat-hint>наприклад.Хірургія</mat-hint>
            </mat-form-field>
          </div>
          <div>
            <mat-form-field appearance="outline">
              <mat-label>Виробник</mat-label>
              <input matInput />
              <mat-hint>наприклад.Bosh</mat-hint>
            </mat-form-field>
          </div>
```

```
          <div>
            <button mat-button color="primary">ШУКАТИ</button>
          </div>
        </div>
      </form>
    </mat-card>
  </div>
</div>
<div [ngStyle]="{'margin-top': '2rem'}">
  <h1 class="section-name">Результати пошуку</h1>
  <div>
    <table
      mat-table
      [dataSource]="dataSource"
    >
    <ng-container matColumnDef="Name">
        <mat-header-cell *matHeaderCellDef class="name">
          Назва
        </mat-header-cell>
        <mat-cell *matCellDef="let inventory" class="name">
          <div>{{inventory.name}}</div>
        </mat-cell>
      </ng-container>
      <ng-container matColumnDef="Manufacturer">
          <mat-header-cell *matHeaderCellDef>
            Виробник
          </mat-header-cell>
          <mat-cell *matCellDef="let inventory">
            <div>{{inventory.manufacturer}}</div>
          </mat-cell>
        </ng-container>
        <ng-container matColumnDef="Date of release">
            <mat-header-cell *matHeaderCellDef>
              Дата виробництва
            </mat-header-cell>
            <mat-cell *matCellDef="let inventory">
              <div>{{inventory.dateOfRelease}}</div>
            </mat-cell>
          </ng-container>
          <ng-container matColumnDef="Location">
              <mat-header-cell *matHeaderCellDef class="address">
                Адреса
              </mat-header-cell>
              <mat-cell        *matCellDef="let        inventory"
class="address">
                <div>{{inventory.location}}</div>
              </mat-cell>
```

```
            </ng-container>
            <ng-container matColumnDef="Count">
                <mat-header-cell *matHeaderCellDef>
                  Кількість
                </mat-header-cell>
                <mat-cell *matCellDef="let inventory">
                  <div>{{inventory.count}}</div>
                </mat-cell>
              </ng-container>
              <ng-container matColumnDef="More">
                  <mat-header-cell *matHeaderCellDef>
                    </mat-header-cell>
                  <mat-cell *matCellDef="let inventory">
                    <div><mat-icon>arrow_right_alt
                      </mat-icon></div>
                  </mat-cell>
                </ng-container>
        <mat-header-row
*matHeaderRowDef="displayedColumns"></mat-header-row>
        <mat-row       *matRowDef="let       row;       columns:
displayedColumns"></mat-row>
    </table>
  </div>
</div>
```

2.6. СЕРВІС ДЛЯ РОБОТИ З КОРИСТУВАЧАМИ

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository, getRepository, DeleteResult } from 'typeorm';
import { UserEntity } from './user.entity';
import {CreateUserDto, LoginUserDto, UpdateUserDto} from './dto';
const jwt = require('jsonwebtoken');
import { SECRET } from '../config';
import { UserRO } from './user.interface';
import { validate } from 'class-validator';
import          {          HttpException          }          from
'@nestjs/common/exceptions/http.exception';
import { HttpStatus } from '@nestjs/common';
import * as argon2 from 'argon2';

@Injectable()
export class UserService {
  constructor(
    @InjectRepository(UserEntity)
    private readonly userRepository: Repository<UserEntity>
  ) {}
```

```
  async findAll(): Promise<UserEntity[]> {
    return await this.userRepository.find();
  }

  async findOne({email, password}: LoginUserDto):
Promise<UserEntity> {
    const user = await this.userRepository.findOne({email});
    if (!user) {
      return null;
    }

    if (await argon2.verify(user.password, password)) {
      return user;
    }

    return null;
  }

  async create(dto: CreateUserDto): Promise<UserRO> {

    // check uniqueness of username/email
    const {username, email, password} = dto;
    const qb = await getRepository(UserEntity)
      .createQueryBuilder('user')
      .where('user.username = :username', { username })
      .orWhere('user.email = :email', { email });

    const user = await qb.getOne();

    if (user) {
      const errors = {username: 'Username and email must be
unique.'};
      throw new HttpException({message: 'Input data validation
failed', errors}, HttpStatus.BAD_REQUEST);

    }

    // create new user
    let newUser = new UserEntity();
    newUser.username = username;
    newUser.email = email;
    newUser.password = password;

    const errors = await validate(newUser);
    if (errors.length > 0) {
      const _errors = {username: 'Userinput is not valid.'};
```

```
      throw new HttpException({message: 'Input data validation
failed', _errors}, HttpStatus.BAD_REQUEST);

    } else {
      const savedUser = await this.userRepository.save(newUser);
      return this.buildUserRO(savedUser);
    }

  }

  async update(id: number, dto: UpdateUserDto):
Promise<UserEntity> {
    let toUpdate = await this.userRepository.findOne(id);
    delete toUpdate.password;
    delete toUpdate.favorites;

    let updated = Object.assign(toUpdate, dto);
    return await this.userRepository.save(updated);
  }

  async delete(email: string): Promise<DeleteResult> {
    return await this.userRepository.delete({ email: email});
  }

  async findById(id: number): Promise<UserRO>{
    const user = await this.userRepository.findOne(id);

    if (!user) {
      const errors = {User: ' not found'};
      throw new HttpException({errors}, 401);
    }

    return this.buildUserRO(user);
  }

  async findByEmail(email: string): Promise<UserRO>{
    const user = await this.userRepository.findOne({email:
email});
    return this.buildUserRO(user);
  }

  public generateJWT(user) {
    let today = new Date();
    let exp = new Date(today);
    exp.setDate(today.getDate() + 60);

    return jwt.sign({
```

```
      id: user.id,
      username: user.username,
      email: user.email,
      exp: exp.getTime() / 1000,
    }, SECRET);
  };

  private buildUserRO(user: UserEntity) {
    const userRO = {
      id: user.id,
      username: user.username,
      email: user.email,
      bio: user.bio,
      token: this.generateJWT(user),
      image: user.image
    };

    return {user: userRO};
  }
}
```

2.7. ФУНКЦІЇ ТРАНСФОРМАЦІЇ HTTP-ВІДПОВІДЕЙ

```
import {PipeTransform, ArgumentMetadata, BadRequestException,
HttpStatus, Injectable} from '@nestjs/common';
import { validate } from 'class-validator';
import { plainToClass } from 'class-transformer';
import { HttpException } from
'@nestjs/common/exceptions/http.exception';

@Injectable()
export class ValidationPipe implements PipeTransform<any> {
  async transform(value, metadata: ArgumentMetadata) {

    if (!value) {
      throw new BadRequestException('No data submitted');
    }

    const { metatype } = metadata;
    if (!metatype || !this.toValidate(metatype)) {
      return value;
    }
    const object = plainToClass(metatype, value);
    const errors = await validate(object);
    if (errors.length > 0) {
```

```
      throw new HttpException({message: 'Input data validation
failed', errors:  this.buildError(errors)},
HttpStatus.BAD_REQUEST);
    }
    return value;
  }

  private buildError(errors) {
    const result = {};
    errors.forEach(el => {
      let prop = el.property;
      Object.entries(el.constraints).forEach(constraint => {
        result[prop + constraint[0]] = `${constraint[1]}`;
      });
    });
    return result;
  }

  private toValidate(metatype): boolean {
    const types = [String, Boolean, Number, Array, Object];
    return !types.find((type) => metatype === type);
  }
}
```

2.8 КЛАС ДЛЯ ПЕРЕВІРКИ ПРАВ ДОСТУПУ КОРИСТУВАЧА НА СЕРВЕРНІЙ ЧАСТИНІ

```
import { HttpException } from
'@nestjs/common/exceptions/http.exception';
import { NestMiddleware, HttpStatus, Injectable } from
'@nestjs/common';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { Request, Response, NextFunction } from 'express';
import * as jwt from 'jsonwebtoken';
import { SECRET } from '../config';
import { UserService } from './user.service';

@Injectable()
export class AuthMiddleware implements NestMiddleware {
  constructor(private readonly userService: UserService) {}

  async use(req: Request, res: Response, next: NextFunction) {
    const authHeaders = req.headers.authorization;
    if (authHeaders && (authHeaders as string).split(' ')[1]) {
      const token = (authHeaders as string).split(' ')[1];
      const decoded: any = jwt.verify(token, SECRET);
      const user = await this.userService.findById(decoded.id);
```

```
      if (!user) {
        throw new HttpException('User not found.',
HttpStatus.UNAUTHORIZED);
      }

      req.user = user.user;
      next();

    } else {
      throw new HttpException('Not authorized.',
HttpStatus.UNAUTHORIZED);
    }
  }
```

Додаток 3

Копії презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

АНАЛІТИЧНО – ІНФОРМАЦІЙНА СИСТЕМА СТВОРЕННЯ КРИТИЧНИХ МЕДИЧНИХ ЗАЯВОК

Виконав: Самойленко Назарій Юрійович

Керівник: старший викладач кафедри ПЗКС, к.т.н. Хіцко Я.В

Київ – 2020

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: Розробити аналітично-інформаційну систему для створення та опрацювання медичних заявок з критично необхідного інвентаря.

Завдання:

1. Проаналізувати предметну область
2. Проаналізувати та вибрати програмні засоби для реалізації проекту
3. Розробити архітектуру додатку та реалізувати додаток у вигляді веб-застосунку
4. Протестувати розроблений додаток

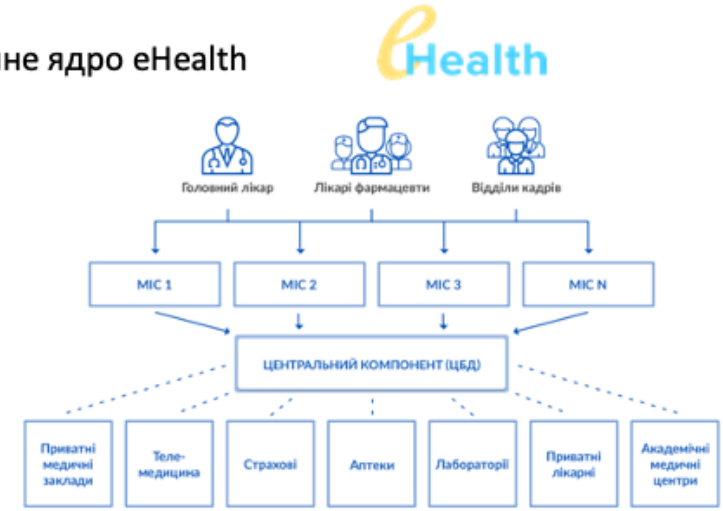
АКТУАЛЬНІСТЬ

Впровадження інформаційних технологій в сферу охорони здоров’я

- Швидкість прийняття рішень
- Полегшення комунікацій
- Відсутність прямих аналогічних рішень

Аналіз предметної області

- Технічне ядро eHealth



Вибір засобів для розроблення проєкту

Angular

NgRX

PostgreSQL

Nest.js

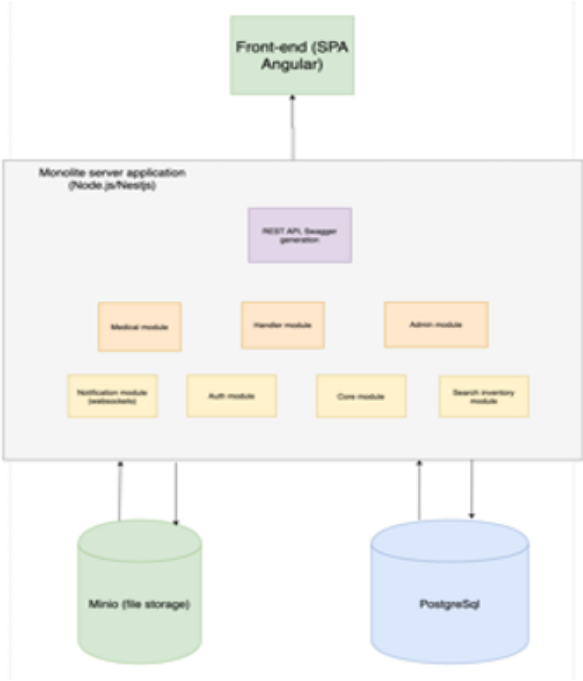
Swagger

TypeORM

TypeScript

Архітектура системи

Модульний монолітний сервер та клієнтський застосунок, що обмінюються даними між собою за рахунок RESTful API.



Розроблені алгоритми. Життєвий цикл заявки

Розроблені алгоритми. Життєвий цикл заявки

Створення заявки

Критичні заявки

Створити заявку Переглянути створені заявки

Перегляд створених заявок

Критичні заявки

Створити заявку **Переглянути створені заявки**

Назва	Номер лікарні	Виробник	Адреса	Опис	Статус
Т-подібна пластина	43-12-42	Астера	м.Київ, вул. Харківське шосе, 121	Опис пластини	Відкрита
Шпиці Кіршрена	43-12-42	Астера	м.Київ, вул. Харківське шосе, 121	Опис шпиців	Відкрита

Пошук інвентаря

Пошук інвентаря

Результати пошуку

Назва	Виробник	Дата виробництва	Адреса	Кількість	
Апарат штучної вентиляції легенів	Біомед	18-06-2016	м. Київ, пров. Дем'ївський 5А	2	→
Апарат штучної вентиляції легенів	Біомед	28-07-2018	м. Київ, вул.Червонофлотська, 26	5	→
Апарат штучної вентиляції легенів	Біомед	18-11-2019	м. Київ, вул. Солом'янська, 17	4	→

Перегляд статистичних даних

Статистика статусів критичних заявок

- Створені: 44
- Закриті: 55
- Підтверджені: 13
- Потребують доопрацювання: 43

Співвідношення створених заявок на додавання інвентаря до підтверджених

Тестування

- проведено ручне тестування
- створені тестові випадки (test - cases)
- протестований графічний інтерфейс користувача на наявність помилок в роботі та User Experience (досвід користувача, наскільки зручно користуватися системою)
- знайдені помилки було виправлено

Висновки

- Проаналізовано програмні рішення (бази даних, серверні застосунки, клієнтські застосунки)
- Розроблено та реалізовано архітектуру веб – додатку
- Розроблено алгоритми повного життєвого циклу заявки з критично необхідного інвентаря
- Система пройшла ручне тестування – виявлені помилки було виправлено
- Унікальність роботи: 95.2%

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

АНАЛІТИЧНО-ІНФОРМАЦІЙНА СИСТЕМА СТВОРЕННЯ КРИТИЧНИХ МЕДИЧНИХ ЗАЯВОК

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Яна ХІЦКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Назарій САМОЙЛЕНКО

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Аналітично-інформаційна система для створення заявок з критично необхідного інвентаря, що складається з серверного і клієнтського застосунку та бази даних. З використанням мови програмування TypeScript, та бібліотек: Angular, Nest.js та СКБД PostgreSQL.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірене наступне:

- функціональна працездатність API системи;
- коректна робота системи авторизації та ідентифікації користувача;
- наявність доступу до бази даних;
- коректна робота системи синхронізації веб-серверу та клієнтського додатку;
- коректна робота клієнтської частини застосунку;
- коректна система відображення помилок;
- зручність та простота роботи з API;
- відповідність дизайну вимогам технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

- функціональне тестування, зокрема на рівні Smoke test, що перевіряє базові функції та Critical path test («тестування 20% функціональності, що використовують 80% користувачів»);
- тестування продуктивності програмного забезпечення при різних навантаженнях користувачів;

- тестування графічного інтерфейсу користувача на наявність помилок в роботі та User Experience (досвід користувача, наскільки зручно користуватися системою).

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконувалася в ручному режимі та за допомогою утиліти Postman, яка дозволяє моделювати HTTP-запити.

Працездатність веб-сервісу перевіряється шляхом:

- динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
- динамічного ручного тестування на відповідність фунціональним вимогам;
- статичного тестування коду;
- тестування при максимальному навантаженні;
- тестування стабільності роботи при різних умовах;
- тестування зручності використання;
- тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

АНАЛІТИЧНО-ІНФОРМАЦІЙНА СИСТЕМА СТВОРЕННЯ КРИТИЧНИХ МЕДИЧНИХ ЗАЯВОК

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Яна ХІЦКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Назарій САМОЙЛЕНКО

2020

ЗМІСТ

1. ОПИС СТРУКТУРИ ПРОГРАМНОЇ СИСТЕМИ

Програмна система розроблена у вигляді веб-додатку, що складається із статичних та динамічних веб-сторінок, тобто таких, вміст яких залишається статичним та формується динамічно відповідно.

Веб-додаток є одномовним, використовується українська мова. Застосунок має панель навігації, для переходу між сторінками сайту. Основними сторінками сайту є наступні:

1. Головна сторінка.
2. Сторінка авторизації користувача.
3. Сторінка створення заявок з критично необхідного медичного інвентаря.
4. Сторінка перегляду заявки.
5. Сторінка створення заявки на додавання інвентаря до бази даних.
6. Сторінка пошуку інвентаря.

Вміст кожної з сторінок залежить від прав доступу авторизованого користувача. Всього в системі передбачено 3 ролі користувачів: Медичний працівник, Обробник Заявок, Адміністратор.

2. ПРОЦЕДУРА АВТОРИЗАЦІЇ КОРИСТУВАЧА

Для отримання доступу до системи користувач має пройти процедуру авторизації. Для цього він використовує вікно авторизації, що наведено на рис. 2.1.

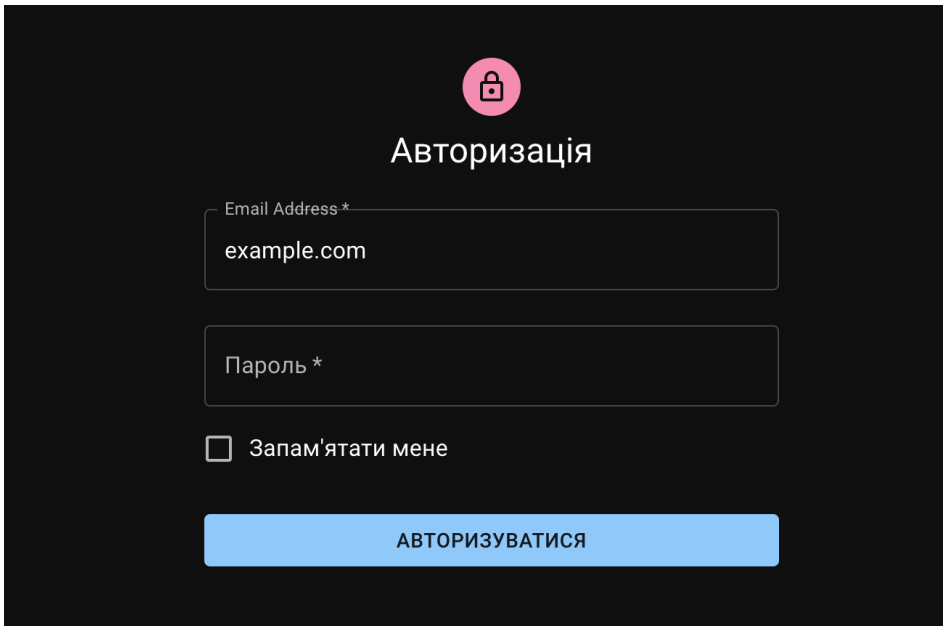


Рис. 2.1 Вікно авторизації користувача

Користувач вводить логін та пароль, після цього дані опрацьовуються на сервері користувач проходить процес ідентифікації, аутентентифікації та авторизації користувача (надані прав доступу). Після цього користувач переходить до головної сторінки.

3. НАВІГАЦІЯ КОРИСТУВАЧА

За допомогою бокової панелі навігації він може переміщуватися між сторінками сайту. Для ідентифікації поточної сторінки користувача на навігаційній панелі використовується смужка індикатор синього кольору зліва від елемента навігації, що наведена на рис. 3.1.

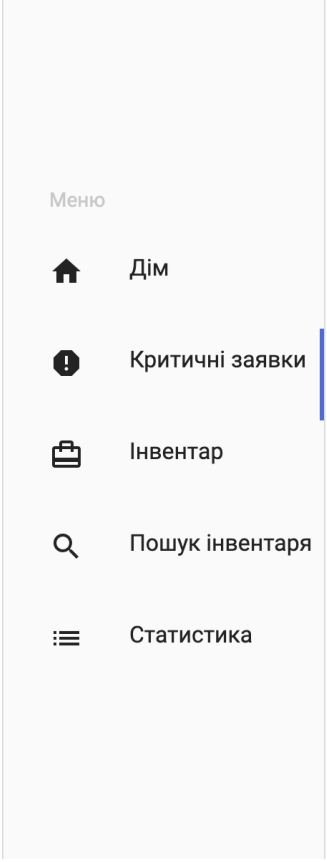


Рис. 3.1 Бокова панель навігації сайту

4. СТВОРЕННЯ ЗАЯВКИ З КРИТИЧНО НЕОБХІДНОГО ІНВЕНТАРЯ

Перейшовши за допомогою навігації до сторінки «Критичні заявки», користувач може створити заявку з необхідного медичного інвентаря. На сторінці є дві вкладки, в одній з них користувач може створити заявку та відправити її, в іншій переглянути список створених ним заявок та відслідкувати їх статус. Для цього йому необхідно заповнити поля форми, що наведена на рис. 4.1, та відправити її.

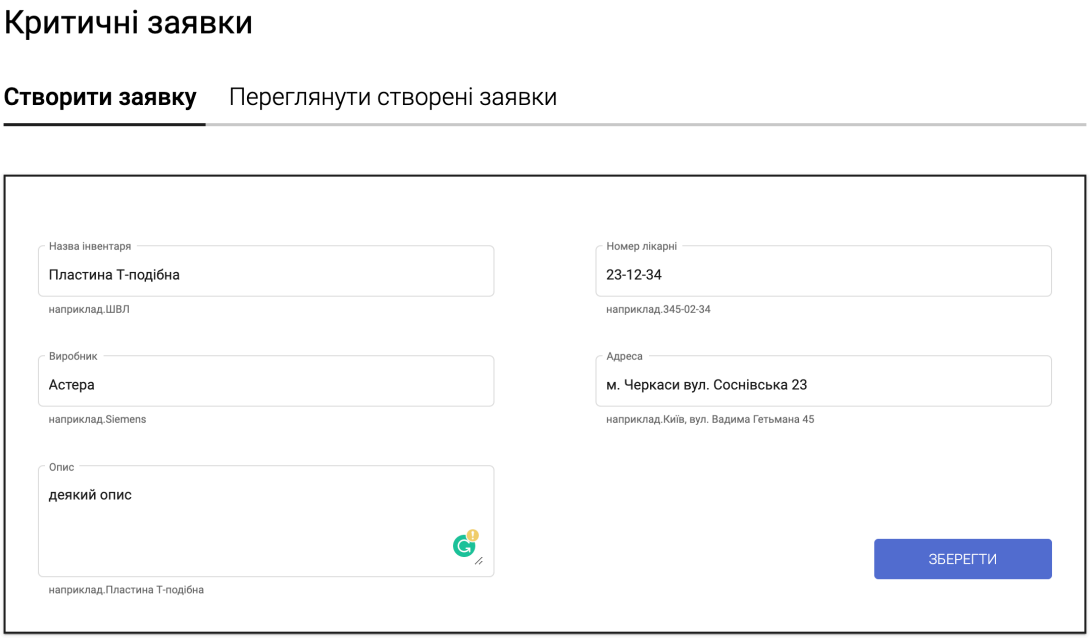


Рис. 4.1 Форма для створення заявки з критично необхідного інвентаря

Перейшовши на вкладку «Переглянути створені заявки», користувач може переглянути створені заявки в таблиці, що наведена на рис. 4.2.

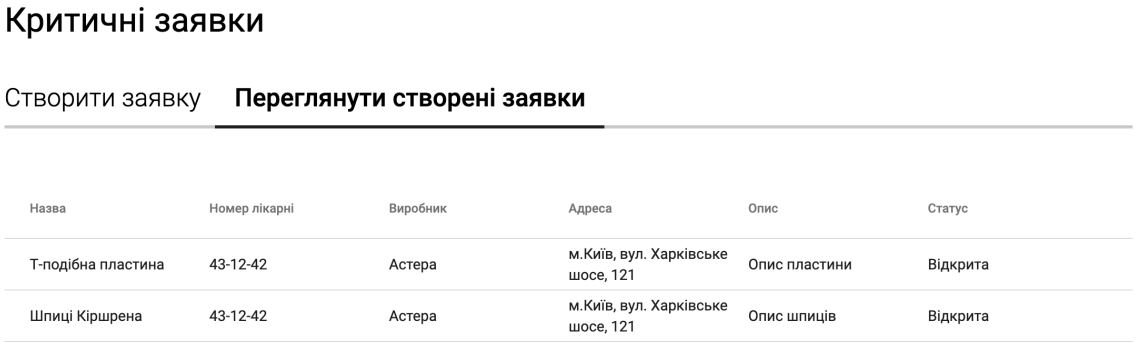


Рис. 4.2 Таблиця заявок з критично необхідного інвентаря

В системі передбачено наступні статуси заявок: відкрита, потребує доопрацювання, підтверджена, закрита.

5. ПЕРЕГЛЯД СТАТИСТИЧНИХ ДАНИХ

Користувач з правами доступу адміністратор має доступ до сторінки перегляду статистичних даних, що накопилися в ході роботи застосунку. На рисунку 5.1 показано кругову діаграму, що показує співвідношення між статусами заявок, що були створені в системі. На рисунку 5.2 показано лінійній графік показує загальну кількість створених заявок у відношенні до місяця, вибравши точку на графіку можна побачити кількість створених заявок у конкретний місяць. На рисунку 5.3 показано графік співвідношення кількості створених заявок на додавання інвентарю до бази даних до успішно доданих, у відповідності до місяця.

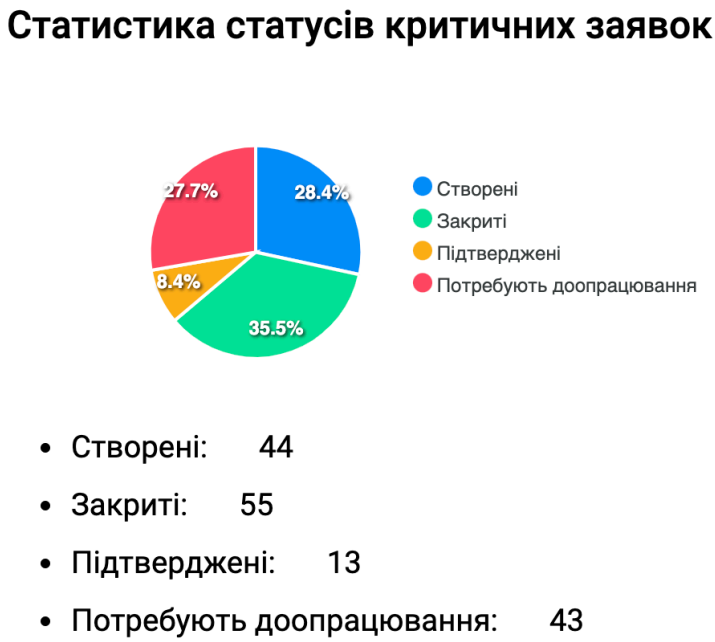


Рис. 5.1 Перегляд статистики статусів заявок з критично необхідного медичного інвентаря

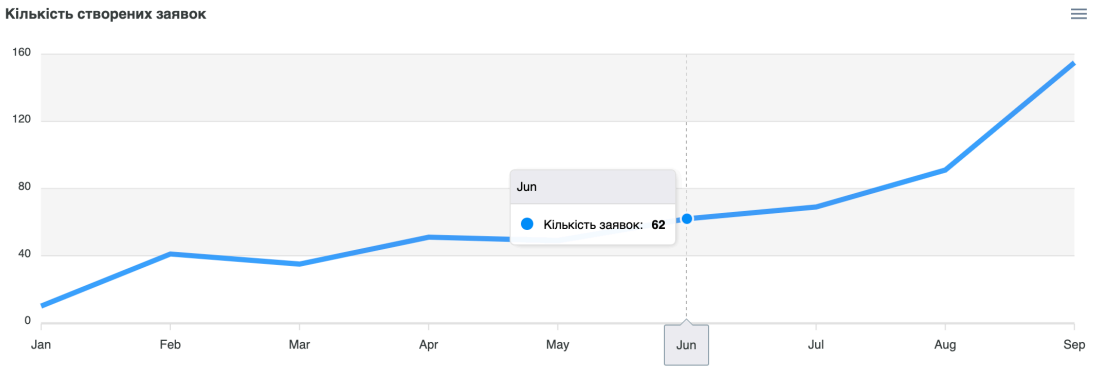


Рис. 5.2 Лінійний графік створених заявок

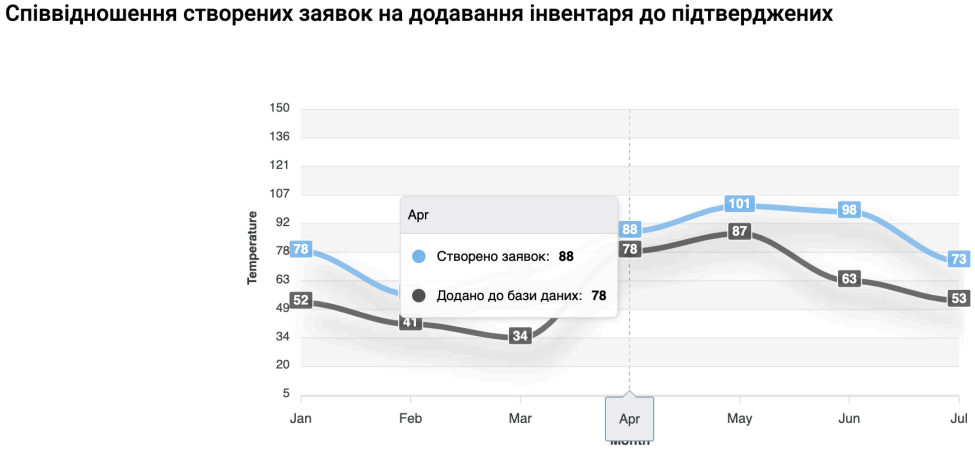


Рис. 5.3 Графік кількості створених та успішних заявок на додавання інвентарю до бази даних